Amazon SageMaker
Developer Guide
# Table of Contents

What Is Amazon SageMaker? .................................................................................................................................................................................. 1  
Amazon SageMaker Features ............................................................................................................................................................................. 1  
Amazon SageMaker Pricing ............................................................................................................................................................................. 2  
Are You a First-time User of Amazon SageMaker? ........................................................................................................................................... 2  
How It Works ................................................................................................................................................................................................. 3  
Machine Learning with Amazon SageMaker ....................................................................................................................................................... 3  
Explore, Analyze, and Process Data ................................................................................................................................................................ 5  
Model Training ......................................................................................................................................................................................................... 5  
Model Deployment ...................................................................................................................................................................................................... 8  
Model Monitoring .................................................................................................................................................................................................. 13  
Programming Model .................................................................................................................................................................................................... 13  
Set Up Amazon SageMaker ............................................................................................................................................................................... 15  
Create an AWS Account .................................................................................................................................................................................................. 15  
Create an IAM Administrator User and Group .................................................................................................................................................. 15  
Onboard to Amazon SageMaker Studio .............................................................................................................................................................. 16  
Onboard using SSO as a Small Team .................................................................................................................................................................... 16  
Onboard using SSO as an Organization .............................................................................................................................................................. 17  
Sign-on using SSO ................................................................................................................................................................................................... 18  
Onboard using IAM .................................................................................................................................................................................................. 18  
Sign-on using IAM .................................................................................................................................................................................................. 19  
Get Started with Amazon SageMaker ............................................................................................................................................................ 20  
Get Started with Studio ......................................................................................................................................................................................... 20  
UI Overview ...................................................................................................................................................................................................................... 20  
Features ....................................................................................................................................................................................................................... 23  
Get Started with the Console ............................................................................................................................................................................. 24  
Step 1: Create an Amazon S3 Bucket ................................................................................................................................................................. 25  
Step 2: Create an Amazon SageMaker Notebook Instance .................................................................................................................................. 26  
Step 3: Create a Jupyter Notebook .................................................................................................................................................................... 27  
Step 4: Download, Explore, and Transform Data ............................................................................................................................................. 28  
Step 5: Train a Model ..................................................................................................................................................................................................... 30  
Step 6: Deploy the Model ................................................................................................................................................................................................... 34  
Step 7: Validate the Model ................................................................................................................................................................................................... 39  
Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications ................................................................. 43  
Step 9: Clean Up ....................................................................................................................................................................................................... 43  
Use Autopilot to Automate Model Development .................................................................................................................................................... 45  
Get Started ...................................................................................................................................................................................................................... 45  
Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio .................................................................................................. 45  
SageMaker Autopilot Notebook Output .............................................................................................................................................................. 46  
Candidate Generation Notebook ........................................................................................................................................................................... 46  
Data Exploration Notebook .................................................................................................................................................................................. 47  
SageMaker Autopilot Problem Types ................................................................................................................................................................... 47  
Linear regression ........................................................................................................................................................................................................... 47  
Binary classification ......................................................................................................................................................................................................... 47  
Multi-class classification .................................................................................................................................................................................................. 47  
Automatic problem type detection ..................................................................................................................................................................... 48  
Prepare and Label Data .................................................................................................................................................................................................. 49  
Use Ground Truth for Labeling ................................................................................................................................................................................................... 49
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy Models</td>
<td>605</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>605</td>
</tr>
<tr>
<td>What do you want to do?</td>
<td>605</td>
</tr>
<tr>
<td>Manage Model Deployments</td>
<td>605</td>
</tr>
<tr>
<td>Deploy Your Own Inference Code</td>
<td>606</td>
</tr>
<tr>
<td>Guide to Amazon SageMaker</td>
<td>606</td>
</tr>
<tr>
<td>Multi-Model Endpoints</td>
<td>606</td>
</tr>
<tr>
<td>Sample Notebooks</td>
<td>607</td>
</tr>
<tr>
<td>How Multi-Model Endpoints Work</td>
<td>607</td>
</tr>
<tr>
<td>Security</td>
<td>607</td>
</tr>
<tr>
<td>Instance Recommendations for Multi-Model Endpoint Deployments</td>
<td>608</td>
</tr>
<tr>
<td>CloudWatch Metrics for Multi-Model Endpoint Deployments</td>
<td>609</td>
</tr>
<tr>
<td>Create a Multi-Model Endpoint</td>
<td>609</td>
</tr>
<tr>
<td>Bring Your Own Container</td>
<td>612</td>
</tr>
<tr>
<td>Invoke a Multi-Model Endpoint</td>
<td>615</td>
</tr>
<tr>
<td>Add or Remove Models</td>
<td>615</td>
</tr>
<tr>
<td>Model Monitor</td>
<td>616</td>
</tr>
<tr>
<td>How It Works</td>
<td>616</td>
</tr>
<tr>
<td>Capture Data</td>
<td>617</td>
</tr>
<tr>
<td>Create a Baseline</td>
<td>619</td>
</tr>
<tr>
<td>Schedule Monitoring Jobs</td>
<td>620</td>
</tr>
<tr>
<td>Interpret Results</td>
<td>623</td>
</tr>
<tr>
<td>Advanced Topics</td>
<td>629</td>
</tr>
<tr>
<td>Inference Pipelines</td>
<td>642</td>
</tr>
<tr>
<td>Sample Notebooks</td>
<td>642</td>
</tr>
<tr>
<td>Process Features with Spark ML and Scikit-learn</td>
<td>643</td>
</tr>
<tr>
<td>Create a Pipeline Model</td>
<td>643</td>
</tr>
<tr>
<td>Real-time Inference</td>
<td>646</td>
</tr>
<tr>
<td>Batch Transform</td>
<td>648</td>
</tr>
<tr>
<td>Logs and Metrics</td>
<td>649</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>654</td>
</tr>
<tr>
<td>Use Batch Transform</td>
<td>656</td>
</tr>
<tr>
<td>Use Batch Transform to Get Inferences from Large Datasets</td>
<td>656</td>
</tr>
<tr>
<td>Speed up a Batch Transform Job</td>
<td>658</td>
</tr>
<tr>
<td>Use Batch Transform to Test Production Variants</td>
<td>658</td>
</tr>
<tr>
<td>Batch Transform Errors</td>
<td>658</td>
</tr>
<tr>
<td>Sample Notebooks</td>
<td>658</td>
</tr>
<tr>
<td>Associate Prediction Results with Input</td>
<td>658</td>
</tr>
<tr>
<td>Compile and Deploy Models with Neo</td>
<td>663</td>
</tr>
<tr>
<td>Sample Notebooks</td>
<td>664</td>
</tr>
<tr>
<td>Compile Models</td>
<td>664</td>
</tr>
<tr>
<td>Deploy Models</td>
<td>669</td>
</tr>
<tr>
<td>Request Inferences</td>
<td>677</td>
</tr>
<tr>
<td>Troubleshoot Errors</td>
<td>677</td>
</tr>
<tr>
<td>Elastic Inference</td>
<td>683</td>
</tr>
</tbody>
</table>
What Is Amazon SageMaker?

Amazon SageMaker is a fully managed machine learning service. With Amazon SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, Amazon SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a single click from the Amazon SageMaker console. Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

Topics
- Amazon SageMaker Features (p. 1)
- Amazon SageMaker Pricing (p. 2)
- Are You a First-time User of Amazon SageMaker? (p. 2)

Amazon SageMaker Features

Amazon SageMaker includes the following features:

**Amazon SageMaker Studio**

An integrated machine learning environment where you can build, train, deploy, and analyze your models all in the same application.

**Amazon SageMaker Ground Truth**

High-quality training datasets by using workers along with machine learning to create labeled datasets.

**Amazon Augmented AI**

Human-in-the-loop reviews

**Amazon SageMaker Studio Notebooks**

The next generation of Amazon SageMaker notebooks that include SSO integration, fast start-up times, and single-click sharing.

**Preprocessing**

Analyze and pre-process data, tackle feature engineering, and evaluate models.

**Amazon SageMaker Experiments**

Experiment management and tracking. You can use the tracked data to reconstruct an experiment, incrementally build on experiments conducted by peers, and trace model lineage for compliance and audit verifications.

**Amazon SageMaker Debugger**

Inspect training parameters and data throughout the training process. Automatically detect and alert users to commonly occurring errors such as parameter values getting too large or small.
Amazon SageMaker Autopilot

Users without machine learning knowledge can quickly build classification and regression models.

Reinforcement Learning

Maximize the long-term reward that an agent receives as a result of its actions.

Batch Transform

Preprocess datasets, run inference when you don't need a persistent endpoint, and associate input records with inferences to assist the interpretation of results.

Amazon SageMaker Model Monitor

Monitor and analyze models in production (endpoints) to detect data drift and deviations in model quality.

Amazon SageMaker Neo

Train machine learning models once, then run anywhere in the cloud and at the edge.

Amazon SageMaker Elastic Inference

Speed up the throughput and decrease the latency of getting real-time inferences.

Amazon SageMaker Pricing

As with other AWS products, there are no contracts or minimum commitments for using Amazon SageMaker. For more information about the cost of using Amazon SageMaker, see Amazon SageMaker Pricing.

Are You a First-time User of Amazon SageMaker?

If you are a first-time user of Amazon SageMaker, we recommend that you do the following:

1. Read How Amazon SageMaker Works (p. 3) – This section provides an overview of Amazon SageMaker, explains key concepts, and describes the core components involved in building AI solutions with Amazon SageMaker. We recommend that you read this topic in the order presented.
2. Read Get Started with Amazon SageMaker (p. 20) – This section explains how to set up your account and create your first Amazon SageMaker notebook instance.
3. Try a model training exercise – This exercise walks you through training your first model. You use training algorithms provided by Amazon SageMaker. For more information, see Get Started with Amazon SageMaker (p. 20).
4. Explore other topics – Depending on your needs, do the following:
   • Submit Python code to train with deep learning frameworks – In Amazon SageMaker, you can use your own training scripts to train models. For information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 443).
   • Use Amazon SageMaker directly from Apache Spark – For information, see Use Apache Spark with Amazon SageMaker (p. 443).
   • Use Amazon AI to train and/or deploy your own custom algorithms – Package your custom algorithms with Docker so you can train and/or deploy them in Amazon SageMaker. See Use Your Own Algorithms or Models with Amazon SageMaker (p. 456) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.
5. See the API Reference (p. 843) – This section describes the Amazon SageMaker API operations.
How Amazon SageMaker Works

Amazon SageMaker is a fully managed service that enables you to quickly and easily integrate machine learning-based models into your applications. This section provides an overview of machine learning and explains how Amazon SageMaker works. If you are a first-time user of Amazon SageMaker, we recommend that you read the following sections in order:

Topics
- Machine Learning with Amazon SageMaker (p. 3)
- Explore, Analyze, and Process Data (p. 5)
- Train a Model with Amazon SageMaker (p. 5)
- Deploy a Model in Amazon SageMaker (p. 8)
- Monitoring a Model in Production (p. 13)
- The Amazon SageMaker Programming Model (p. 13)

How It Works: Next Topic

Machine Learning with Amazon SageMaker (p. 3)

Machine Learning with Amazon SageMaker

This section describes a typical machine learning workflow and summarizes how you accomplish those tasks with Amazon SageMaker.

In machine learning, you “teach” a computer to make predictions, or inferences. First, you use an algorithm and example data to train a model. Then you integrate your model into your application to generate inferences in real time and at scale. In a production environment, a model typically learns from millions of example data items and produces inferences in hundreds to less than 20 millisecond.

The following diagram illustrates the typical workflow for creating a machine learning model:
As the diagram illustrates, you typically perform the following activities:

1. **Generate example data**—To train a model, you need example data. The type of data that you need depends on the business problem that you want the model to solve (the inferences that you want the model to generate). For example, suppose that you want to create a model to predict a number given an input image of a handwritten digit. To train such a model, you need example images of handwritten numbers.

   Data scientists often spend a lot of time exploring and preprocessing, or "wrangling," example data before using it for model training. To preprocess data, you typically do the following:

   a. **Fetch the data**—You might have in-house example data repositories, or you might use datasets that are publicly available. Typically, you pull the dataset or datasets into a single repository.

   b. **Clean the data**—To improve model training, inspect the data and clean it as needed. For example, if your data has a country name attribute with values United States and US, you might want to edit the data to be consistent.

   c. **Prepare or transform the data**—To improve performance, you might perform additional data transformations. For example, you might choose to combine attributes. If your model predicts the conditions that require de-icing an aircraft, instead of using temperature and humidity attributes separately, you might combine those attributes into a new attribute to get a better model.

   In Amazon SageMaker, you preprocess example data in a Jupyter notebook on your notebook instance. You use your notebook to fetch your dataset, explore it, and prepare it for model training. For more information, see Explore, Analyze, and Process Data (p. 5). For more information about preparing data in AWS Marketplace, see data preparation.

2. **Train a model**—Model training includes both training and evaluating the model, as follows:

   - **Training the model**—To train a model, you need an algorithm. The algorithm you choose depends on a number of factors. For a quick, out-of-the-box solution, you might be able to use one of the algorithms that Amazon SageMaker provides. For a list of algorithms provided by Amazon SageMaker and related considerations, see Use Amazon SageMaker Built-in Algorithms (p. 220).

   You also need compute resources for training. Depending on the size of your training dataset and how quickly you need the results, you can use resources ranging from a single general-purpose instance to a distributed cluster of GPU instances. For more information, see Train a Model with Amazon SageMaker (p. 5).

   - **Evaluating the model**—After you’ve trained your model, you evaluate it to determine whether the accuracy of the inferences is acceptable. In Amazon SageMaker, you use either the AWS SDK for Python (Boto) or the high-level Python library that Amazon SageMaker provides to send requests to the model for inferences.

      You use a Jupyter notebook in your Amazon SageMaker notebook instance to train and evaluate your model.

3. **Deploy the model**—You traditionally re-engineer a model before you integrate it with your application and deploy it. With Amazon SageMaker hosting services, you can deploy your model independently, decoupling it from your application code. For more information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 8).

   Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, collect "ground truth," and evaluate the model to identify drift. You then increase the accuracy of your inferences by updating your training data to include the newly collected ground truth. You do this by retraining the model with the new dataset. As more and more example data becomes available, you continue retraining your model to increase accuracy.
Explore, Analyze, and Process Data

Before using a dataset to train a model, data scientists typically explore, analyze, and preprocess it. For example, in one of the exercises in this guide, you use the MNIST dataset, a commonly available dataset of handwritten numbers, for model training. Before you begin training, you transform the data into a format that is more efficient for training. For more information, see Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 29).

To preprocess data use one of the following methods:

- Use a Jupyter notebook on an Amazon SageMaker notebook instance. You can also use the notebook instance to do the following:
  - Write code to create model training jobs
  - Deploy models to Amazon SageMaker hosting
  - Test or validate your models

  For more information, see Use Amazon SageMaker Notebook Instances (p. 201)

- You can use a model to transform data by using Amazon SageMaker batch transform. For more information, see Step 6.2: Deploy the Model with Batch Transform (p. 37).

Amazon SageMaker Processing enables running jobs to preprocess and postprocess data, perform feature engineering, and evaluate models on Amazon SageMaker easily and at scale. When combined with the other critical machine learning tasks provided by Amazon SageMaker, such as training and hosting, Processing provides you with the benefits of a fully managed machine learning environment, including all the security and compliance support built into Amazon SageMaker. With Processing, you have the flexibility to use the built-in data processing containers or to bring your own containers and submit custom jobs to run on managed infrastructure. After you submit a job, Amazon SageMaker launches the compute instances, processes and analyzes the input data, and releases the resources upon completion. For more information, see Process Data and Evaluate Models (p. 189).

- For information about how to run your own data processing scripts, Data Processing and Model Evaluation with Scikit-Learn (p. 190).
- For information about how to build your own processing container to run scripts, see Build Your Own Processing Container (p. 191).
The area labeled Amazon SageMaker highlights the two components of Amazon SageMaker: model training and model deployment.

To train a model in Amazon SageMaker, you create a training job. The training job includes the following information:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 222).

You have the following options for a training algorithm:

- **Use an algorithm provided by Amazon SageMaker**—Amazon SageMaker provides training algorithms. If one of these meets your needs, it’s a great out-of-the-box solution for quick model
training. For a list of algorithms provided by Amazon SageMaker, see Use Amazon SageMaker Built-in Algorithms (p. 220). To try an exercise that uses an algorithm provided by Amazon SageMaker, see Get Started with Amazon SageMaker (p. 20).

- **Use Amazon SageMaker Debugger**—to inspect training parameters and data throughout the training process when working with the TensorFlow, PyTorch, and Apache MXNet learning frameworks or the XGBoost algorithm. Debugger automatically detects and alerts users to commonly occurring errors such as parameter values getting too large or small. For more information about using Debugger, see Amazon SageMaker Debugger (p. 516). Debugger sample notebooks are available at Amazon SageMaker Debugger Samples.

- **Use Apache Spark with Amazon SageMaker**—Amazon SageMaker provides a library that you can use in Apache Spark to train models with Amazon SageMaker. Using the library provided by Amazon SageMaker is similar to using Apache Spark MLLib. For more information, see Use Apache Spark with Amazon SageMaker (p. 443).

- **Submit custom code to train with deep learning frameworks**—You can submit custom Python code that uses TensorFlow or Apache MXNet for model training. For more information, see Use TensorFlow with Amazon SageMaker (p. 452) and Use Apache MXNet with Amazon SageMaker (p. 453).

- **Use your own custom algorithms**—Put your code together as a Docker image and specify the registry path of the image in an Amazon SageMaker CreateTrainingJob API call. For more information, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 456).

- **Use an algorithm that you subscribe to from AWS Marketplace**—For information, see Find and Subscribe to Algorithms and Model Packages on AWS Marketplace (p. 502).

After you create the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

You can create a training job with the Amazon SageMaker console or the API. For information about creating a training job with the API, see the CreateTrainingJob (p. 931) API.

When you create a training job with the API, Amazon SageMaker replicates the entire dataset on ML compute instances by default. To make Amazon SageMaker replicate a subset of the data on each ML compute instance, you must set the S3DataDistributionType field to ShardedByS3Key. You can set this field using the low-level SDK. For more information, see S3DataDistributionType in S3DataSource (p. 1501).

**Important**
To prevent your algorithm container from contending for memory, you should reserve some memory for Amazon SageMaker critical system processes on your ML compute instances. If the algorithm container is allowed to use memory needed for system processes, it can trigger a system failure.
How It Works: Next Topic
Deploy a Model in Amazon SageMaker (p. 8)

Deploy a Model in Amazon SageMaker

After you train your model, you can deploy it to get predictions in one of two ways:

- To set up a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics
- Deploy a Model on Amazon SageMaker Hosting Services (p. 8)
- Get Inferences for an Entire Dataset with Batch Transform (p. 11)
- Validate a Machine Learning Model (p. 12)

Deploy a Model on Amazon SageMaker Hosting Services

Amazon SageMaker also provides model hosting services for model deployment, as shown in the following diagram. Amazon SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences.
Deploying a model using Amazon SageMaker hosting services is a three-step process:

1. **Create a model in Amazon SageMaker**—By creating a model, you tell Amazon SageMaker where it can find the model components. This includes the S3 path where the model artifacts are stored and the Docker registry path for the image that contains the inference code. In subsequent deployment steps, you specify the model by name. For more information, see the `CreateModel (p. 902)` API.

2. **Create an endpoint configuration for an HTTPS endpoint**—You specify the name of one or more models in production variants and the ML compute instances that you want Amazon SageMaker to launch to host each production variant.

   When hosting models in production, you can configure the endpoint to elastically scale the deployed ML compute instances. For each production variant, you specify the number of ML compute instances that you want to deploy. When you specify two or more instances, Amazon SageMaker launches them in multiple Availability Zones. This ensures continuous availability. Amazon SageMaker manages deploying the instances. For more information, see the `CreateEndpointConfig (p. 878)` API.

3. **Create an HTTPS endpoint**—Provide the endpoint configuration to Amazon SageMaker. The service launches the ML compute instances and deploys the model or models as specified in the configuration. For more information, see the `CreateEndpoint (p. 875)` API. To get inferences from
the model, client applications send requests to the Amazon SageMaker Runtime HTTPS endpoint. For more information about the API, see the InvokeEndpoint (p. 1260) API.

Note
When you create an endpoint, Amazon SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. For a list of instance types that Amazon SageMaker hosting service supports, see AWS Service Limits. For a list of the sizes of the storage volumes that Amazon SageMaker attaches to each instance, see Host Instance Storage Volumes (p. 710).

To increase a model’s accuracy, you might choose to save the user’s input data and ground truth, if available, as part of the training data. You can then retrain the model periodically with a larger, improved training dataset.

Best Practices for Deploying Models on Amazon SageMaker Hosting Services

When hosting models using Amazon SageMaker hosting services, consider the following:

• Typically, a client application sends requests to the Amazon SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.

• You can deploy a model trained with Amazon SageMaker to your own deployment target. To do that, you need to know the algorithm-specific format of the model artifacts that were generated by model training. For more information about output formats, see the section corresponding to the algorithm you are using in Training Data Formats (p. 229).

• You can deploy multiple variants of a model to the same Amazon SageMaker HTTPS endpoint. This is useful for testing variations of a model in production. For example, suppose that you’ve deployed a model into production. You want to test a variation of the model by directing a small amount of traffic, say 5%, to the new model. To do this, create an endpoint configuration that describes both variants of the model. You specify the ProductionVariant in your request to the CreateEndPointConfig. For more information, see ProductionVariant (p. 1484).

• You can configure a ProductionVariant to use Application Auto Scaling. For information about configuring automatic scaling, see Automatically Scale Amazon SageMaker Models (p. 694).

• You can modify an endpoint without taking models that are already deployed into production out of service. For example, you can add new model variants, update the ML Compute instance configurations of existing model variants, or change the distribution of traffic among model variants. To modify an endpoint, you provide a new endpoint configuration. Amazon SageMaker implements the changes without any downtime. For more information see, UpdateEndpoint (p. 1233) and UpdateEndpointWeightsAndCapacities (p. 1235).

• Changing or deleting model artifacts or changing inference code after deploying a model produces unpredictable results. If you need to change or delete model artifacts or change inference code, modify the endpoint by providing a new endpoint configuration. Once you provide the new endpoint configuration, you can change or delete the model artifacts corresponding to the old endpoint configuration.
• If you want to get inferences on entire datasets, consider using batch transform as an alternative to hosting services. For information, see Get Inferences for an Entire Dataset with Batch Transform (p. 11)

How It Works: Next Topic
Validate a Machine Learning Model (p. 12)

Get Inferences for an Entire Dataset with Batch Transform

To get inferences for an entire dataset, use batch transform. With batch transform, you create a batch transform job using a trained model and the dataset, which must be stored in Amazon S3. Amazon SageMaker saves the inferences in an S3 bucket that you specify when you create the batch transform job. Batch transform manages all of the compute resources required to get inferences. This includes launching instances and deleting them after the batch transform job has completed. Batch transform manages interactions between the data and the model with an object within the instance node called an agent.

Use batch transform when you:

• Want to get inferences for an entire dataset and index them to serve inferences in real time
• Don’t need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences
• Don’t need the subsecond latency that Amazon SageMaker hosted endpoints provide

You can also use batch transform to preprocess your data before using it to train a new model or generate inferences.

The following diagram shows the workflow of a batch transform job:

To perform a batch transform, create a batch transform job using either the Amazon SageMaker console or the API. Provide the following:

• The path to the S3 bucket where you’ve stored the data that you want to transform.
• The compute resources that you want Amazon SageMaker to use for the transform job. Compute resources are machine learning (ML) compute instances that are managed by Amazon SageMaker.
Validating Models

• The path to the S3 bucket where you want to store the output of the job.
• The name of the Amazon SageMaker model that you want to use to create inferences. You must use a model that you have already created either with the `CreateModel (p. 902)` operation or the console.

The following is an example of what a dataset file might look like.

```
An example of input file content:
Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-
  AttributeM
Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-
  AttributeM
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-
  AttributeM
...  
RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-
  AttributeM
```

A record is a single input data unit. For information about how to delimit records for batch transform jobs, see `SplitType` in `TransformInput (p. 1536)`.

For an example of how to use batch transform, see Step 6.2: Deploy the Model with Batch Transform (p. 37).

How It Works: Next Topic

Validate a Machine Learning Model (p. 12)

Validate a Machine Learning Model

After training a model, evaluate it to determine whether its performance and accuracy allow you to achieve your business goals. You might generate multiple models using different methods and evaluate each. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You might consider whether your model needs to be more sensitive than specific (or vice versa).

You can evaluate your model using historical data (offline) or live data:

• **Offline testing**—Use historical, not live, data to send requests to the model for inferences.

  Deploy your trained model to an alpha endpoint, and use historical data to send inference requests to it. To send the requests, use a Jupyter notebook in your Amazon SageMaker notebook instance and either the AWS SDK for Python (Boto) or the high-level Python library provided by Amazon SageMaker.

• **Online testing with live data**—Amazon SageMaker supports deploying multiple models (called production variants) to a single Amazon SageMaker endpoint. You configure the production variants so that a small portion of the live traffic goes to the model that you want to validate. For example, you might choose to send 10% of the traffic to a model variant for evaluation. After you are satisfied with the model's performance, you can route 100% traffic to the updated model.

  For more information, see articles and books about how to evaluate models, for example, *Evaluating Machine Learning Models*.

Options for offline model evaluation include:
• **Validating using a “holdout set”**—Machine learning practitioners often set aside a part of the data as a "holdout set." They don't use this data for model training.

With this approach, you evaluate how well your model provides inferences on the holdout set. You then assess how effectively the model generalizes what it learned in the initial training, as opposed to using model "memory." This approach to validation gives you an idea of how often the model is able to infer the correct answer.

In some ways, this approach is similar to teaching elementary school students. First, you provide them with a set of examples to learn, and then test their ability to generalize from their learning. With homework and tests, you pose problems that were not included in the initial learning and determine whether they are able to generalize effectively. Students with perfect memories could memorize the problems, instead of learning the rules.

Typically, the holdout dataset is of 20-30% of the training data.

• **k-fold validation**—In this validation approach, you split the example dataset into $k$ parts. You treat each of these parts as a holdout set for $k$ training runs, and use the other $k-1$ parts as the training set for that run. You produce $k$ models using a similar process, and aggregate the models to generate your final model. The value $k$ is typically in the range of 5-10.

### How It Works: Next Topic

*The Amazon SageMaker Programming Model (p. 13)*

## Monitoring a Model in Production

After you deploy a model into your production environment, Amazon SageMaker Model Monitor can be used to continuously monitor the quality of Amazon SageMaker machine learning models. It enables developers to set alerts for when there are deviations in the model quality, such as data drift and anomalies. Early and pro-active detection of these deviations enables you to take corrective actions. For more information, see *Amazon SageMaker Model Monitor (p. 616)*.

### The Amazon SageMaker Programming Model

Amazon SageMaker provides APIs that you can use to create and manage notebook instances and train and deploy models. For more information, see *API Reference (p. 843)*.

Making API calls directly from code is cumbersome, and requires you to write code to authenticate your requests. Amazon SageMaker provides the following alternatives:

• **Use the Amazon SageMaker console**—With the console, you don't write any code. You use the console UI to start model training or deploy a model. The console works well for simple jobs, where you use a built-in training algorithm and you don't need to preprocess training data.

• **Modify the example Jupyter notebooks**—Amazon SageMaker provides several Jupyter notebooks that train and deploy models using specific algorithms and datasets. Start with a notebook that has a suitable algorithm and modify it to accommodate your data source and specific needs.
• **Write model training and inference code from scratch**—Amazon SageMaker provides both an AWS SDK and a high-level Python library that you can use in your code to start model training jobs and deploy the resulting models.

• **The high-level Python library**—The Python library simplifies model training and deployment. In addition to authenticating your requests, the library abstracts platform specifics by providing simple methods and default parameters. For example:
  
  - To deploy your model, you call only the `deploy()` method. The method creates an Amazon SageMaker model, an endpoint configuration, and an endpoint.
  
  - If you use a custom framework script for model training, you call the `fit()` method. The method creates a .gzip file of your script, uploads it to an Amazon S3 location, and then runs it for model training, and other tasks. For more information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 443).

• **The AWS SDK**—The SDKs provide methods that correspond to the Amazon SageMaker API (see Actions (p. 843)). Use the SDKs to programmatically start a model training job and host the model in Amazon SageMaker. SDK clients authenticate your requests by using your access keys, so you don't need to write authentication code. They are available in multiple languages and platforms. For more information, see SDKs.

In Get Started with Amazon SageMaker (p. 20), you train and deploy a model using an algorithm provided by Amazon SageMaker. That exercise shows how to use both of these libraries. For more information, see Get Started with Amazon SageMaker (p. 20).

• **Integrate Amazon SageMaker into your Apache Spark workflow**—Amazon SageMaker provides a library for calling its APIs from Apache Spark. With it, you can use Amazon SageMaker-based estimators in an Apache Spark pipeline. For more information, see Use Apache Spark with Amazon SageMaker (p. 443).

**How It Works: Next Topic**

Get Started with Amazon SageMaker (p. 20)
Set Up Amazon SageMaker

In this section, you sign up for an AWS account, create an IAM admin user, and create an SSO Organization account.

If you're new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 3).

Topics
- Create an AWS Account (p. 15)
- Create an IAM Administrator User and Group (p. 15)
- Onboard to Amazon SageMaker Studio (p. 16)

Create an AWS Account

In this section, you sign up for an AWS account. If you already have an AWS account, skip this step.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Amazon SageMaker. You are charged only for the services that you use.

To create an AWS account
2. Follow the online instructions.
   - Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Write down your AWS account ID because you'll need it for the next task.

Create an IAM Administrator User and Group

When you create an AWS account, you get a single sign-in identity that has complete access to all of the AWS services and resources in the account. This identity is called the AWS account root user. Signing in to the AWS console using the email address and password that you used to create the account gives you complete access to all of the AWS resources in your account.

We strongly recommend that you not use the root user for everyday tasks, even the administrative ones. Instead, adhere to the Create Individual IAM Users, an AWS Identity and Access Management (IAM) administrator user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

To create an administrator user
- Create an administrator user in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the IAM User Guide.
  
  Note
  We assume that you use administrator user credentials for the exercises and procedures in this guide. If you choose to create and use another IAM user, grant that user minimum permissions. For more information, see Authenticating with Identities (p. 729).
Onboard to Amazon SageMaker Studio

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

To get started with Amazon SageMaker Studio and Amazon SageMaker Notebooks, you must complete the Studio onboard process. Users can choose between two modes of authentication: AWS Single Sign-On (AWS SSO) and AWS Identity and Access Management (IAM).

**Note**
Amazon SageMaker Studio is available only in the US East (Ohio) Region, us-east-2. The link on the Amazon SageMaker Management Console to access Studio appears only when the default AWS Region in the Console is set to US East (Ohio). To change the region, use the drop-down list located at the top right of the screen next to your AWS account number.

For AWS SSO, you can sign up as an individual or small team, or as an organization. Organizations have the option to batch load their users from their existing AWS SSO organization.

Some of the benefits of using AWS SSO over IAM:

- Allows organizations to batch load their users.
- Your data scientists or developers have a unique sign-on URL and use their corporate credentials. They don't have to interact with the AWS Management Console to use Amazon SageMaker Studio or to run the organization's Amazon SageMaker Studio Notebooks.

**Topics**
- Onboard to Amazon SageMaker Studio using AWS SSO as a Small Team (p. 16)
- Onboard to Amazon SageMaker Studio using AWS SSO as an Organization (p. 17)
- Sign-on to Amazon SageMaker Studio using AWS SSO (p. 18)
- Onboard to Amazon SageMaker Studio using IAM (p. 18)
- Sign-on to Amazon SageMaker Studio using IAM (p. 19)

---

Onboard to Amazon SageMaker Studio using AWS SSO as a Small Team

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

The following procedure describes how to onboard to Amazon SageMaker Studio for a single user or a team. For information on onboarding as an AWS SSO organization, see Onboard to Amazon SageMaker Studio using AWS SSO as an Organization (p. 17).

**To onboard to Amazon SageMaker Studio using AWS SSO**

1. Open the **Amazon SageMaker Management Console**. Select **Amazon SageMaker Studio** at the top left of the screen and then the **Amazon SageMaker Studio** page opens.
2. Under **Get started**, select **Start with SSO** and then the **Get started** page opens.
3. If you don't have an AWS SSO account, the page displays a **Create a user** section. Enter your email address and name, and then choose **Create**.

   Studio creates an AWS SSO account for you and assigns you as a user.
If you do have an AWS SSO account, this page isn't displayed and you should continue with the next step.

4. The Get started page opens. Under Permission, open the drop-down list, select Create a new role, and then the Create an IAM role dialog opens.

5. Under S3 buckets you specify, specify additional buckets, besides those listed, that users of your notebooks can access. If you don't want to add additional buckets, choose None. Next, select Create role.

   Studio creates a new IAM role with the AmazonSageMakerFullAccess and AmazonSageMaker-ExecutionPolicy policies attached. This role is applied to every user in your AWS SSO account.

6. On the Get started page, select Submit, and then the Amazon SageMaker Studio page opens.

   Note
   If you cancel at this step, the previously created AWS SSO account and user are still created. To delete the account and user, you must use the AWS SSO Management Console.

   Wait for the Status to change to InService and then the Assign users button is enabled.

7. Select Assign users and then the Assign users page opens with the previously created user listed.

8. To grant the user access to Amazon SageMaker Studio, select the checkbox next to the user.

9. Select Assign and then Studio sends an email to the user with an invitation to join AWS Single Sign-On. The Amazon SageMaker Studio page opens.

To sign-on to Studio, see Sign-on to Amazon SageMaker Studio using AWS SSO (p. 18).

Onboard to Amazon SageMaker Studio using AWS SSO as an Organization

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

The following procedure describes how to onboard to Amazon SageMaker Studio for an AWS SSO organization. For information on onboarding as a single user or team, see Onboard to Amazon SageMaker Studio using AWS SSO as a Small Team (p. 16).

To create an AWS Single Sign-On organization account and add users

1. Open the Amazon SageMaker Management Console. Select Amazon SageMaker Studio at the top left of the screen and then the Amazon SageMaker Studio page opens.

2. Under Get started, select Start with SSO and then the Get started page opens.

3. Under Permission, open the drop-down list and select Create new role. Amazon SageMaker creates a new IAM role with the AmazonSageMakerFullAccess policy attached. This role will be applied to every user in your AWS SSO account.

   Select Create account and then the Amazon SageMaker Studio page opens.

4. Select Assign users and then the Assign users page opens.

5. The users in your AWS SSO account are listed. Select the users that you want to grant access to Amazon SageMaker Studio.

   Select Save and then the Amazon SageMaker Studio page reopens with the selected users listed.

To sign-on to Studio, see Sign-on to Amazon SageMaker Studio using AWS SSO (p. 18).
Sign-on to Amazon SageMaker Studio using AWS SSO

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

When you are granted access to Amazon SageMaker Studio, Amazon SageMaker sends you an email with the subject “Invitation to join AWS Single Sign-On”.

To sign-on to Studio as a first time user

1. In the invitation email, select Accept invitation, and then an AWS web page opens.
2. Choose and confirm your password, and then select Update user.
3. On the success notification, select Continue, and then the Single Sign-On page opens.
4. Select Amazon SageMaker Studio, and then wait for the Amazon SageMaker Studio landing page to open.

To sign-on as a return user, you have two choices

- On the invitation email, select the link under Your User portal URL.
- Sign in to the Amazon SageMaker Console and then select Amazon SageMaker Studio at the top left of the page. The Amazon SageMaker Studio page opens. Select the Amazon SageMaker Studio address link under the Summary section.

Onboard to Amazon SageMaker Studio using IAM

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

The following procedure describes how to onboard to Amazon SageMaker using AWS Identity and Access Management (IAM). For information on onboarding using AWS SSO, see Onboard to Amazon SageMaker Studio using AWS SSO as a Small Team (p. 16) or Onboard to Amazon SageMaker Studio using AWS SSO as an Organization (p. 17).

This procedure directs you through the onboarding process for Amazon SageMaker Studio using IAM.

To onboard to Studio using IAM

1. Open the Amazon SageMaker Management Console. Select Amazon SageMaker Studio at the top left of the screen and then the Amazon SageMaker Studio page opens.
2. Under Get started, choose Start with IAM, and then the Get started page opens.
3. Under Permission, open the drop-down list, select Create a new role, and then the Create an IAM role dialog opens.
4. Under S3 buckets you specify, specify additional buckets, besides those listed, that users of your notebooks can access. If you don't want to add additional buckets, choose None. Next, select Create role.

   Amazon SageMaker creates a new IAM role with the AmazonSageMakerFullAccess policy attached.

5. Choose Submit and then the Amazon SageMaker Studio landing page opens. Wait for the Status to change to InService and then the Add user profile button is enabled.
6. Choose **Add user profile** and then the **Add user profile** page opens.

7. Under **User profile**, enter a user profile name and choose an execution role. The name is limited to upper- and lower-case letters, digits, and the hyphen, '-', character. Keep the default **Execution role**.

   Choose **Submit** and then the Amazon SageMaker Studio landing page opens with the new user profile listed.

8. Repeat the previous two steps to add more user profiles.

9. To start using Amazon SageMaker Studio, see **Sign-on to Amazon SageMaker Studio using IAM** (p. 19).

**Sign-on to Amazon SageMaker Studio using IAM**

---

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

This procedure describes the IAM sign-on process for Amazon SageMaker Studio. You must onboard to Studio before you can sign-on. For more information, see **Onboard to Amazon SageMaker Studio using IAM** (p. 18).

**To sign-on to Studio using IAM**

1. Open the Amazon SageMaker Management Console. Select **Amazon SageMaker Studio** at the top left of the screen and then the **Amazon SageMaker Studio** page opens showing a list of user profiles.

2. Choose the **User profiles** tab and then choose your profile. The **Open Amazon SageMaker Studio** button is enabled.

3. Select **Open Amazon SageMaker Studio** and then wait for the Amazon SageMaker Studio landing page to open.
Get Started with Amazon SageMaker

Topics

- Get Started with Amazon SageMaker Studio (p. 20)
- Get Started with the Amazon SageMaker Console (p. 24)

Get Started with Amazon SageMaker Studio

Amazon SageMaker Studio is an integrated development environment (IDE) for machine learning that lets you build, train, debug, deploy, and monitor your machine learning models. Studio provides all the tools you need to take your models from experimentation to production while boosting your productivity. In a single unified visual interface, customers can

- Write and execute code in Jupyter notebooks
- Build and train machine learning models
- Deploy the models and monitor the performance of their predictions
- Track and debug the machine learning experiments

The following sections provide an overview of the user interface and a description of Studio’s main features.

For information on the onboarding steps to sign-on to Amazon SageMaker Studio, see Onboard to Amazon SageMaker Studio (p. 16).

For a tutorial that demonstrates the basic features of Amazon SageMaker Studio, see Track and Evaluate a Model Training Experiment (p. 506).

Topics

- Amazon SageMaker Studio UI Overview (p. 20)
- Amazon SageMaker Studio Features (p. 23)

Amazon SageMaker Studio UI Overview

Amazon SageMaker Studio extends the JupyterLab interface. Previous users of JupyterLab will notice the similarity of the user interface, including the workspace. Studio adds many additions to the interface. The most prominent additions are detailed in the following sections. For an overview of the basic JupyterLab interface, see The JupyterLab Interface.

The following image shows the initial screen when you sign-on to Amazon SageMaker Studio.
At the top of the screen is the menu bar. At the left of the screen is the left sidebar which contains icons for different file and resource browsers. At the right of the screen is the right sidebar, represented by the gear icon, which displays contextual preference settings. At the bottom of the screen is the status bar.

The main work area is divided horizontally into two panes. The left pane is the file and resource browser. The right pane contains one or more tabs for resources such as notebooks, terminals, metrics, and graphs.

**Topics**
- Left Sidebar (p. 21)
- SageMaker Experiment List (p. 22)
- SageMaker Endpoint List (p. 22)
- Using Notebooks (p. 22)
- View a Trial’s Properties and Components (p. 23)

**Left Sidebar**

The left sidebar includes the following icons. When you hover over an icon, a tooltip displays the icon name. When you select an icon, the file and resource browser displays the described functionality. For hierarchical entries, a selectable breadcrumb at the top of the browser shows your location.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="File icon" /></td>
<td>File browser.</td>
</tr>
<tr>
<td><img src="image" alt="Terminal icon" /></td>
<td>Running terminals and kernels.</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image" alt="Git" /></td>
<td>Git.</td>
</tr>
<tr>
<td><img src="image" alt="Commands" /></td>
<td>Commands (Ctrl + Shift + C). The majority of the menu commands are available here.</td>
</tr>
<tr>
<td><img src="image" alt="SageMaker Experiment List" /></td>
<td>SageMaker Experiment List. Displays a list of experiments, trials, or trial components. Includes a button to create an Autopilot experiment.</td>
</tr>
<tr>
<td><img src="image" alt="SageMaker Endpoint List" /></td>
<td>SageMaker Endpoint List. Selected when you choose <strong>Deploy and monitor models</strong> on the Studio landing page.</td>
</tr>
<tr>
<td><img src="image" alt="Open tabs" /></td>
<td>Open tabs</td>
</tr>
</tbody>
</table>

### SageMaker Experiment List

Displays a list of experiments, trials, or trial components, and allows you to create an Amazon SageMaker Autopilot experiment. The following primary actions are available:

- To create an Autopilot experiment, choose **Create Experiment**.
- To display the list of trials that are part of an experiment, double-click the experiment.
- To display the list of trial components that make up a trial, double-click the trial.
- To open a tab in the work area that describes a component, double-click the component.
- To compare experiments or trials, multi-select the items, right-click one of the selections, and then choose **Open in trial component list**.

A selectable breadcrumb above the header shows your position in the hierarchy and allows you to navigate to a higher level.

### SageMaker Endpoint List

To open a tab that shows a description of an endpoint, double click the endpoint.

### Using Notebooks

To open a notebook, follow these steps:

1. In the left sidebar, select the **File Browser** icon.
2. At the top of the file browser, select the **Up arrow** icon, and then a **File Upload** dialog opens. Browse to and select the notebook and any accompanying files, and then choose **Open**.
3. Double-click the uploaded notebook file to open the notebook in a new tab.
View a Trial's Properties and Components

To display the list of trials that are part of an experiment, choose the SageMaker Experiment List icon and then double-click the experiment.

There are two ways to view a trial's properties and components. Each method displays similar information in different formats, as well as unique information as noted in each item below.

- Right click one of the trials and then choose Open in trial component list. A new tab opens that displays a list of the trial's components.
  
  Use this view to deploy a model.

- Double-click one of the trials and a list of the trial's components is displayed. Double-click one of the components in the list and a new tab opens that describes each component.

Amazon SageMaker Studio Features

Amazon SageMaker Studio includes the following features.

Topics

- Amazon SageMaker Studio Notebooks (p. 23)
- Amazon SageMaker Experiments (p. 23)
- Amazon SageMaker Autopilot (p. 24)
- Amazon SageMaker Debugger (p. 24)
- Amazon SageMaker Model Monitor (p. 24)

Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio Notebooks is the next generation of Amazon SageMaker notebooks. These notebooks include the following new features:

- AWS Single Sign-On (AWS SSO) integration
- Fast start-up times
- Ability to share notebooks with a single click

For more information, see Use Amazon SageMaker Notebooks (p. 196).

Note

Because Amazon SageMaker Studio Notebooks is in preview, visual elements of Amazon SageMaker Studio might be impacted.

Amazon SageMaker Experiments

Amazon SageMaker provides experiment management and tracking. Users can organize their experiments and artifacts in a centralized location using a structured organization scheme.

An experiment is a collection of machine learning iterations called trials. A trial is a set of steps called trial components. A trial takes a combination of inputs such as a dataset, an algorithm, and parameters, and produces specific outputs such as a model, metrics, and checkpoints.

Experiment tracking enables both Amazon SageMaker automated tracking of model training, tuning, and evaluation jobs, and API-enabled tracking of experiments done locally on Amazon SageMaker notebooks.
Customers can use the tracked data to reconstruct an experiment, incrementally build on experiments conducted by peers, and trace model lineage for compliance and audit verifications.

For more information, see Manage Machine Learning with Amazon SageMaker Experiments (p. 504).

**Amazon SageMaker Autopilot**

Amazon SageMaker Autopilot provides automatic machine learning that allows users without machine learning knowledge to quickly build classification and regression models. Users only need to provide a tabular dataset and select the target column to predict. Autopilot automatically explores machine learning solutions with different combinations of data preprocessors, algorithms, and algorithm parameters, to find the best model.

When a user runs an Autopilot job, Amazon SageMaker creates an experiment for the job and then creates a trial for each combination and stores all data and results. After the best model is determined, the user can drill down to view each trial and see which features had the most influence on the result.

For more information, see Use Amazon SageMaker Autopilot to Automate Model Development (p. 45).

**Amazon SageMaker Debugger**

Amazon SageMaker Debugger provides full visibility into the model training process by enabling the inspection of all the training parameters and data throughout the training process.

Debugger provides a visual interface to analyze debug data and visual indicators about potential anomalies in the data.

Debugger automatically detects and alerts users to commonly occurring errors such as parameter values getting too large or small. Users can extend Debugger to detect new classes of errors that are specific to their model.

For more information, see Amazon SageMaker Debugger (p. 516).

**Amazon SageMaker Model Monitor**

Amazon SageMaker Model Monitor is a tool for the monitoring and analysis of models in production (Amazon SageMaker endpoints). Model Monitor offers a framework-agnostic analysis.

Machine learning models are typically trained and evaluated using historical data. After they are deployed in production, the quality of their predictions can degrade over time due to model drift. Model drift is when the distribution of the data sent to the models for predictions varies from the distribution of data used during training.

Model Monitor continuously monitors and analyzes the prediction requests. Model Monitor can store this data and use built-in statistical rules to detect common issues such as outliers in data and data drift.

For more information, see Amazon SageMaker Model Monitor (p. 616).

**Get Started with the Amazon SageMaker Console**

The best way to learn how to use Amazon SageMaker is to create, train, and deploy a simple machine learning model. To do this, you need the following:

- A dataset. You use the MNIST (Modified National Institute of Standards and Technology database) dataset of images of handwritten, single digit numbers. This dataset provides a training set of 50,000...
example images of handwritten single-digit numbers, a validation set of 10,000 images, and a test dataset of 10,000 images. You provide this dataset to the algorithm for model training. For more information about the MNIST dataset, see MNIST Dataset.

- An algorithm. You use the XGBoost algorithm provided by Amazon SageMaker to train the model using the MNIST dataset. During model training, the algorithm assigns example data of handwritten numbers into 10 clusters: one for each number, 0 through 9. For more information about the algorithm, see XGBoost Algorithm (p. 422).

You also need a few resources for storing your data and running the code in this exercise:

- An Amazon Simple Storage Service (Amazon S3) bucket to store the training data and the model artifacts that Amazon SageMaker creates when it trains the model.
- An Amazon SageMaker notebook instance to prepare and process data and to train and deploy a machine learning model.
- A Jupyter notebook to use with the notebook instance to prepare your training data and train and deploy the model.

In this exercise, you learn how to create all of the resources that you need to create, train, and deploy a model.

**Important**
For model training, deployment, and validation, you can use either of the following:

- The high-level Amazon SageMaker Python SDK
- The AWS SDK for Python (Boto 3)

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. This exercise provides code examples for both libraries. If you’re a first-time Amazon SageMaker user, we recommend that you use the Amazon SageMaker Python SDK. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

If you’re new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 3) before starting this exercise.

**Topics**

- Step 1: Create an Amazon S3 Bucket (p. 25)
- Step 2: Create an Amazon SageMaker Notebook Instance (p. 26)
- Step 3: Create a Jupyter Notebook (p. 27)
- Step 4: Download, Explore, and Transform the Training Data (p. 28)
- Step 5: Train a Model (p. 30)
- Step 6: Deploy the Model to Amazon SageMaker (p. 34)
- Step 7: Validate the Model (p. 39)
- Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications (p. 43)
- Step 9: Clean Up (p. 43)

**Step 1: Create an Amazon S3 Bucket**

Training a model produces the following

- The model training data
- Model artifacts, which Amazon SageMaker generates during model training
You save these in an Amazon Simple Storage Service (Amazon S3) bucket: You can store datasets that you use as your training data and model artifacts that are the output of a training job in a single bucket or in two separate buckets. For this exercise and others in this guide, one bucket is sufficient. If you already have S3 buckets, you can use them, or you can create new ones.

To create a bucket, follow the instructions in Create a Bucket in the Amazon Simple Storage Service Console User Guide. Include sagemaker in the bucket name. For example, sagemaker-datatime.

**Note**
Amazon SageMaker needs permission to access these buckets. You grant permission with an IAM role, which you create in the next step when you create an Amazon SageMaker notebook instance. This IAM role automatically gets permissions to access any bucket that has sagemaker in the name. It gets these permissions through the AmazonSageMakerFullAccess policy, which Amazon SageMaker attaches to the role. If you add a policy to the role that grants the SageMaker service principal S3FullAccess permission, the name of the bucket does not need to contain sagemaker.

**Next Step**

**Step 2: Create an Amazon SageMaker Notebook Instance**

An Amazon SageMaker notebook instance is a fully managed machine learning (ML) Amazon Elastic Compute Cloud (Amazon EC2) compute instance that runs the Jupyter Notebook App. You use the notebook instance to create and manage Jupyter notebooks that you can use to prepare and process data and to train and deploy machine learning models. For more information, see Explore, Analyze, and Process Data (p. 5).

**Note**
If necessary, you can change the notebook instance settings, including the ML compute instance type, later.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information (if a field is not mentioned, leave the default values):
   a. For **Notebook instance name**, type a name for your notebook instance.
   b. For **Instance type**, choose ml.t2.medium. This is the least expensive instance type that notebook instances support, and it suffices for this exercise.
   c. For **IAM role**, choose **Create a new role**, then choose **Create role**.
   d. Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries.

**Next Step**

**Step 3: Create a Jupyter Notebook (p. 27).
Step 3: Create a Jupyter Notebook

You can create a Jupyter notebook in the notebook instance you created in Step 2: Create an Amazon SageMaker Notebook Instance (p. 26), and create a cell that gets the IAM role that your notebook needs to run Amazon SageMaker APIs and specifies the name of the Amazon S3 bucket that you will use to store the datasets that you use for your training data and the model artifacts that a Amazon SageMaker training job outputs.

To create a Jupyter notebook

1. Open the notebook instance.
   b. Open the Notebook Instances, and then open the notebook instance you created by choosing either Open Jupyter for classic Jupyter view or Open JupyterLab for JupyterLab view next to the name of the notebook instance.

   **Note**
   If you see Pending to the right of the notebook instance in the Status column, your notebook is still being created. The status will change to InService when the notebook is ready for use.

2. Create a notebook.
   a. If you opened the notebook in Jupyter classic view, on the Files tab, choose New, and conda_python3. This preinstalled environment includes the default Anaconda installation and Python 3.
   b. If you opened the notebook in JupyterLab view, on the File menu, choose New, and then choose Notebook. For Select Kernel, choose conda_python3. This preinstalled environment includes the default Anaconda installation and Python 3.

3. In the Jupyter notebook, choose File and Save as, and name the notebook.

4. Copy the following Python code and paste it into the first cell in your notebook. Add the name of the S3 bucket that you created in Set Up Amazon SageMaker (p. 15), and run the code. The get_execution_role function retrieves the IAM role you created when you created your notebook instance.

   ```python
   import os
   import boto3
   import re
   import copy
   import time
   from time import gmtime, strftime
   from sagemaker import get_execution_role

   role = get_execution_role()
   region = boto3.Session().region_name
   bucket='bucket-name' # Replace with your s3 bucket name
   prefix = 'sagemaker/xgboost-mnist' # Used as part of the path in the bucket where you store data
   bucket_path = 'https://s3-{}.amazonaws.com/{}/'.format(region,bucket) # The URL to access the bucket
   ```

Next Step

Step 4: Download, Explore, and Transform the Training Data (p. 28)
Step 4: Download, Explore, and Transform the Training Data

Download the MNIST dataset to your notebook instance, review the data, transform it, and upload it to your S3 bucket.

You transform the data by changing its format from `numpy.array` to comma-separated values (CSV). The XGBoost Algorithm (p. 422) expects input in either the LIBSVM or CSV format. LIBSVM is an open source machine learning library. In this exercise, you use CSV format because it's simpler.

Topics
- Step 4.1: Download the MNIST Dataset (p. 28)
- Step 4.2: Explore the Training Dataset (p. 28)
- Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 29)

Step 4.1: Download the MNIST Dataset

To download the MNIST dataset, copy and paste the following code into the notebook and run it:

```python
%%time
import pickle, gzip, urllib.request, json
import numpy as np

# Load the dataset
urllib.request.urlretrieve("http://deeplearning.net/data/mnist/mnist.pkl.gz", "mnist.pkl.gz")
with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
print(train_set[0].shape)
```

The code does the following:
1. Downloads the MNIST dataset (`mnist.pkl.gz`) from the MNIST Database website to your notebook.
2. Unzips the file and reads the following datasets into the notebook's memory:
   - `train_set` – You use these images of handwritten numbers to train a model.
   - `valid_set` – The XGBoost Algorithm (p. 422) uses these images to evaluate the progress of the model during training.
   - `test_set` – You use this set to get inferences to test the deployed model.

Next Step
- Step 4.2: Explore the Training Dataset (p. 28)

Step 4.2: Explore the Training Dataset

Typically, you explore training data to determine what you need to clean up and which transformations to apply to improve model training. For this exercise, you don't need to clean up the MNIST dataset.

To explore the dataset
- Type the following code in a cell in your notebook and run the cell to display the first 10 images in `train_set`: 

%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (2,10)
for i in range(0, 10):
    img = train_set[0][i]
    label = train_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
plt.show()

train_set contains the following structures:

- train_set[0] – Contains images.

The code uses the matplotlib library to get and display the first 10 images from the training dataset.

Next Step

Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 29)

Step 4.3: Transform the Training Dataset and Upload It to Amazon S3

The XGBoost Algorithm (p. 422) expects comma-separated values (CSV) for its training input. The format of the training dataset is numpy.array. Transform the dataset from numpy.array format to the CSV format. Then upload it to the Amazon S3 bucket that you created in Step 1: Create an Amazon S3 Bucket (p. 25)

To convert the dataset to CSV format and upload it

- Type the following code into a cell in your notebook and then run the cell.
After it converts the dataset to the CSV format, the code uploads the CSV file to the S3 bucket.

**Next Step**

**Step 5: Train a Model (p. 30)**

**Step 5: Train a Model**

To train, deploy, and validate a model in Amazon SageMaker, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). (You can also use the console, but for this exercise, you will use the notebook instance and one of the SDKs.) This exercise provides code examples for each library.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you’re a first-time Amazon SageMaker user, we recommend that you use it to train, deploy, and validate the model. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

**Topics**

- Choose the Training Algorithm (p. 31)
- Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 31)
- Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 32)
Choose the Training Algorithm

To choose the right algorithm for your model, you typically follow an evaluation process. For this exercise, you use the XGBoost Algorithm (p. 422) provided by Amazon SageMaker, so no evaluation is required. For information about choosing algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 220).

Create and Run a Training Job (Amazon SageMaker Python SDK)

The Amazon SageMaker Python SDK includes the sagemaker.estimator.Estimator estimator. You can use this class, in the sagemaker.estimator module, with any algorithm. For more information, see https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator.

To run a model training job (Amazon SageMaker Python SDK)

1. Import the Amazon SageMaker Python SDK and get the XGBoost container.

```python
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

2. Download the training and validation data from the Amazon S3 location where you uploaded it in Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 29), and set the location where you store the training output.

```python
train_data = 's3://{}/{}/{}'.format(bucket, prefix, 'train')
validation_data = 's3://{}/{}/{}'.format(bucket, prefix, 'validation')
s3_output_location = 's3://{}/{}/{}'.format(bucket, prefix, 'xgboost_model_sdk')
print(train_data)
```

3. Create an instance of the sagemaker.estimator.Estimator class.

```python
xgb_model = sagemaker.estimator.Estimator(container,
                                           role,
                                           train_instance_count=1,
                                           train_instance_type='ml.m4.xlarge',
                                           train_volume_size = 5,
                                           output_path=s3_output_location,
                                           sagemaker_session=sagemaker.Session())
```

In the constructor, you specify the following parameters:

- **role** – The AWS Identity and Access Management (IAM) role that Amazon SageMaker can assume to perform tasks on your behalf (for example, reading training results, called model artifacts, from the S3 bucket and writing training results to Amazon S3). This is the role that you got in Step 3: Create a Jupyter Notebook (p. 27).
- **train_instance_count** and **train_instance_type** – The type and number of ML compute instances to use for model training. For this exercise, you use only a single training instance.
- **train_volume_size** – The size, in GB, of the Amazon Elastic Block Store (Amazon EBS) storage volume to attach to the training instance. This must be large enough to store training data if you use File mode (File mode is the default).
- **output_path** – The path to the S3 bucket where Amazon SageMaker stores the training results.
• **sagemaker_session** – The session object that manages interactions with Amazon SageMaker APIs and any other AWS service that the training job uses.

4. Set the hyperparameter values for the XGBoost training job by calling the `set_hyperparameters` method of the estimator. For a description of XGBoost hyperparameters, see XGBoost Hyperparameters (p. 425).

```python
xgb_model.set_hyperparameters(
    max_depth = 5,
    eta = .2,
    gamma = 4,
    min_child_weight = 6,
    silent = 0,
    objective = "multi:softmax",
    num_class = 10,
    num_round = 10)
```

5. Create the training channels to use for the training job. For this example, we use both train and validation channels.

```python
train_channel = sagemaker.session.s3_input(train_data, content_type='text/csv')
valid_channel = sagemaker.session.s3_input(validation_data, content_type='text/csv')
data_channels = {'train': train_channel, 'validation': valid_channel}
```

6. To start model training, call the estimator's `fit` method.

```python
xgb_model.fit(inputs=data_channels, logs=True)
```

    This is a synchronous operation. The method displays progress logs and waits until training completes before returning. For more information about model training, see Train a Model with Amazon SageMaker (p. 5).

    Model training for this exercise can take up to 15 minutes.

**Next Step**

**Step 6: Deploy the Model to Amazon SageMaker (p. 34)**

**Create and Run a Training Job (AWS SDK for Python (Boto 3))**

To train a model, Amazon SageMaker uses the `CreateTrainingJob` (p. 931) API. The AWS SDK for Python (Boto 3) provides the corresponding `create_training_job` method.

When using this method, you provide the following information:

- The training algorithm – Specify the registry path of the Docker image that contains the training code. For the registry paths for the algorithms provided by Amazon SageMaker, see Common Parameters for Built-In Algorithms (p. 222).
- Algorithm-specific hyperparameters – Specify algorithm-specific hyperparameters to influence the final quality of the model. For information, see XGBoost Hyperparameters (p. 425).
- The input and output configuration – Provide the S3 bucket where training data is stored and where Amazon SageMaker saves the results of model training (the model artifacts).

**To run a model training job (AWS SDK for Python (Boto 3))**

1. Import the `get_image_url` utility function Amazon SageMaker Python SDK and get the location of the XGBoost container.
import sagemaker

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'xgboost')

2. Set up the training information for the job. You pass this information when you call `create_training_job`. For more information about the information that you need to send to a training job, see the section called “CreateTrainingJob” (p. 931).

```python
# Ensure that the train and validation data folders generated above are reflected in the "InputDataConfig" parameter below.
common_training_params = {
    "AlgorithmSpecification": {
        "TrainingImage": container,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": bucket_path + "/* prefix */xgboost"
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m4.xlarge",
        "VolumeSizeInGB": 5
    },
    "HyperParameters": {
        "max_depth": "5",
        "eta": "0.2",
        "gamma": "4",
        "min_child_weight": "6",
        "silent": "0",
        "objective": "multi:softmax",
        "num_class": "10",
        "num_round": "10"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": bucket_path + "/* prefix */train/",
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "text/csv",
            "CompressionType": "None"
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": bucket_path + "/* prefix */validation/",
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "text/csv",
            "Compressio
```
3. Name your training job, and finish configuring the parameters that you send to it.

```python
#training job params
training_job_name = 'xgboost-mnist' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Job name is:"; training_job_name)

training_job_params = copy.deepcopy(common_training_params)
training_job_params['TrainingJobName'] = training_job_name
training_job_params['ResourceConfig']['InstanceCount'] = 1
```

4. Call `create_training_job` to start the training job, and wait for it to complete. If the training job fails, print the reason that it failed.

```python
%%time
region = boto3.Session().region_name
sm = boto3.Session().client('sagemaker')
sm.create_training_job(**training_job_params)
status = sm.describe_training_job(TrainingJobName=training_job_name)['TrainingJobStatus']
print(status)
sm.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=training_job_name)
status = sm.describe_training_job(TrainingJobName=training_job_name)['TrainingJobStatus']
print("Training job ended with status: " + status)
if status == 'Failed':
    message = sm.describe_training_job(TrainingJobName=training_job_name)['FailureReason']
    print('Training failed with the following error: {}\n'.format(message))
    raise Exception('Training job failed')
```

You now have a trained model. Amazon SageMaker stores the resulting artifacts in your S3 bucket.

**Next Step**

**Step 6: Deploy the Model to Amazon SageMaker (p. 34)**

**Step 6: Deploy the Model to Amazon SageMaker**

To get predictions, deploy your model. The method you use depends on how you want to generate inferences:

- To get one inference at a time in real time, set up a persistent endpoint using Amazon SageMaker hosting services.
- To get inferences for an entire dataset, use Amazon SageMaker batch transform.

**Topics**

- **Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35)**
- **Step 6.2: Deploy the Model with Batch Transform (p. 37)**
Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services

To deploy a model in Amazon SageMaker, hosting services, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). This exercise provides code examples for both libraries.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you're a first-time Amazon SageMaker user, we recommend that you use it. For more information, see https://sagemaker.readthedocs.io/en/stable/overview.html.

**Topics**

- Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK) (p. 35)
- Deploy the Model to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3)) (p. 35)

**Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)**

Deploy the model that you trained in Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 31) by calling the `deploy` method of the `sagemaker.estimator.Estimator` object. This is the same object that you used to train the model. When you call the `deploy` method, specify the number and type of ML instances that you want to use to host the endpoint.

```python
xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                                  instance_type='ml.m4.xlarge',
                                  )
```

The `deploy` method creates the deployable model, configures the Amazon SageMaker hosting services endpoint, and launches the endpoint to host the model. For more information, see https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator.deploy.

It also returns a `sagemaker.predictor.RealTimePredictor` object, which you can use to get inferences from the model. For information, see https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor.

**Next Step**

Step 7: Validate the Model (p. 39)

**Deploy the Model to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3)).**

Deploying a model using the AWS SDK for Python (Boto 3) is a three-step process:

1. Create a model in Amazon SageMaker – Send a `CreateModel` (p. 902) request to provide information such as the location of the S3 bucket that contains your model artifacts and the registry path of the image that contains inference code.
2. Create an endpoint configuration – Send a `CreateEndpointConfig` (p. 878) request to provide the resource configuration for hosting. This includes the type and number of ML compute instances to launch to deploy the model.
3. Create an endpoint – Send a `CreateEndpoint` (p. 875) request to create an endpoint. Amazon SageMaker launches the ML compute instances and deploys the model. Amazon SageMaker returns an endpoint. Applications can send requests for inference to this endpoint.
To deploy the model (AWS SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 27) and run the cell.

1. Create a deployable model by identifying the location of model artifacts and the Docker image that contains the inference code.

```python
model_name = training_job_name + '-mod'

info = sm.describe_training_job(TrainingJobName=training_job_name)
model_data = info['ModelArtifacts']['S3ModelArtifacts']
print(model_data)

primary_container = {
    'Image': container,
    'ModelDataUrl': model_data
}

create_model_response = sm.create_model(
    ModelName = model_name,
    ExecutionRoleArn = role,
    PrimaryContainer = primary_container)

print(create_model_response['ModelArn'])
```

2. Create an Amazon SageMaker endpoint configuration by specifying the ML compute instances that you want to deploy your model to.

```python
endpoint_config_name = 'DEMO-XGBoostEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_config_name)

create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialVariantWeight':1,
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

3. Create an Amazon SageMaker endpoint.

```python
import time

endpoint_name = 'DEMO-XGBoostEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_name)

create_endpoint_response = sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)

print(create_endpoint_response['EndpointArn'])

resp = sm.describe_endpoint(EndpointName=endpoint_name)
status = resp['EndpointStatus']
print("Status: " + status)

while status=='Creating':
    time.sleep(60)
    resp = sm.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']
```
print("Status: " + status)
print("Arn: " + resp['EndpointArn'])
print("Status: " + status)

This code continuously calls the describe_endpoint command in a while loop until the endpoint either fails or is in service, and then prints the status of the endpoint. When the status changes to InService, the endpoint is ready to serve inference requests.

Next Step

Step 7: Validate the Model (p. 39)

Step 6.2: Deploy the Model with Batch Transform

To get inference for an entire dataset, use batch transform. Amazon SageMaker stores the results in Amazon S3.

For information about batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 11). For an example that uses batch transform, see the batch transform sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_batch_transform/introduction_to_batch_transform.

Topics

• Deploy a Model with Batch Transform (Amazon SageMaker High-level Python Library) (p. 37)
• Deploy a Model with Batch Transform (SDK for Python (Boto 3)) (p. 38)

Deploy a Model with Batch Transform (Amazon SageMaker High-level Python Library)

The following code creates a sagemaker.transformer.Transformer object from the model that you trained in Create and Run a Training Job (Amazon SageMaker Python SDK) (p. 31). Then it calls that object's transform method to create a transform job. When you create the sagemaker.transformer.Transformer object, you specify the number and type of ML instances to use to perform the batch transform job, and the location in Amazon S3 where you want to store the inferences.

Paste the following code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 27) and run the cell.

```python
# The location of the test dataset
batch_input = 's3://{}/{}test/examples'.format(bucket, prefix)

# The location to store the results of the batch transform job
batch_output = 's3://{}/{}batch-inference'.format(bucket, prefix)

transformer = xgb_model.transformer(instance_count=1, instance_type='ml.m4.xlarge',
                                      output_path=batch_output)

transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv',
                       split_type='Line')

transformer.wait()


Next Step
Step 7: Validate the Model (p. 39)

Deploy a Model with Batch Transform (SDK for Python (Boto 3))

To run a batch transform job, call the `create_transform_job` method using the model that you trained in Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 32).

To create a batch transform job (SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 27) and run the cell.

1. Name the batch transform job and specify where the input data (the test dataset) is stored and where to store the job’s output.

   ```python
   batch_job_name = 'xgboost-mnist-batch' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
   batch_input = 's3://{}/{}test/examples'.format(bucket, prefix)
   print(batch_input)
   batch_output = 's3://{}/{}batch-inference'.format(bucket, prefix)
   ```

2. Configure the parameters that you pass when you call the `create_transform_job` method.

   ```python
   request = {
   "TransformJobName": batch_job_name,
   "ModelName": model_name,
   "BatchStrategy": "MultiRecord",
   "TransformOutput": {
   "S3OutputPath": batch_output
   },
   "TransformInput": {
   "DataSource": {
   "S3DataSource": {
   "S3DataType": "S3Prefix",
   "S3Uri": batch_input
   },
   "ContentType": "text/csv",
   "SplitType": "Line",
   "CompressionType": "None"
   },
   "TransformResources": {
   "InstanceType": "ml.m4.xlarge",
   "InstanceCount": 1
   }
   }
   ```

   For more information about the parameters, see the section called “CreateTransformJob” (p. 939).

3. Call the `create_transform_job` method, passing in the parameters that you configured in the previous step. Then call the `describe_transform_job` method in a loop until it completes.

   Paste the following code in a cell in the Jupyter notebook you created in Step 3: Create a Jupyter Notebook (p. 27) and run the cell.

   ```python
   sm.create_transform_job(**request)
   while(True):
   response = sm.describe_transform_job(TransformJobName=batch_job_name)
   status = response['TransformJobStatus']
   ```
if status == 'Completed':
    print("Transform job ended with status: " + status)
    break
if status == 'Failed':
    message = response['FailureReason']
    print("Transform failed with the following error: {}".format(message))
    raise Exception('Transform job failed')
print("Transform job is still in status: " + status)
time.sleep(30)

Next Step

Step 7: Validate the Model (p. 39)

Step 7: Validate the Model

Now that you have trained and deployed a model in Amazon SageMaker, validate it to ensure that it generates accurate predictions on new data. That is, on data that is different from the data that the model was trained on. For this, use the test dataset that you created in Step 4: Download, Explore, and Transform the Training Data (p. 28).

Topics
- Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services (p. 39)
- Step 7.2: Validate a Model Deployed with Batch Transform (p. 42)

Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services

If you deployed a model to Amazon SageMaker hosting services in Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35), you now have an endpoint that you can invoke to get inferences in real time. To validate the model, invoke the endpoint with example images from the test dataset and check whether the inferences you get match the actual labels of the images.

Topics
- Validate a Model Deployed to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK) (p. 39)
- Validate a Model Deployed to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3)) (p. 40)

Validate a Model Deployed to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)

To validate the model by using the Amazon SageMaker Python SDK, use the sagemaker.predictor.RealTimePredictor object that you created in Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK) (p. 35). For information, see https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor.

To validate the model (Amazon SageMaker Python SDK)

1. Download the test data from Amazon S3.

```python
s3 = boto3.resource('s3')
```
Step 7: Validate the Model

2. Plot the first 10 images from the test dataset with their labels.

```python
%matplotlib inline
for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
plt.show()
```

3. To get inferences for the first 10 examples in the test dataset, call the `predict` method of the `sagemaker.predictor.RealTimePredictor` object.

```python
with open('test_data', 'r') as f:
    for j in range(0,10):
        single_test = f.readline()
        result = xgb_predictor.predict(single_test)
        print(result)
```

To see if the model is making accurate predictions, check the output from this step against the numbers that you plotted in the previous step.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

Step 9: Clean Up (p. 43)

Validate a Model Deployed to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3))

To use the AWS SDK for Python (Boto 3) to validate the model, call the `invoke_endpoint` method. This method corresponds to the `InvokeEndpoint` (p. 1260) API provided by the Amazon SageMaker runtime.
To validate the model (AWS SDK for Python (Boto 3))

1. Download the test data from Amazon S3.

```python
s3 = boto3.resource('s3')
test_key = "/{}/test/examples".format(prefix)
s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```python
%matplotlib inline
for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
plt.show()
```

3. Get the Amazon SageMaker runtime client, which provides the `invoke_endpoint` method.

```python
runtime_client = boto3.client('runtime.sagemaker')
```

4. Get inferences from the first 10 examples in the test dataset by calling `invoke_endpoint`.

```python
with open('test_data', 'r') as f:
    for i in range(0,10):
        single_test = f.readline()
        response = runtime_client.invoke_endpoint(EndpointName = endpoint_name,
                                                  ContentType = 'text/csv',
                                                  Body = single_test)
        result = response['Body'].read().decode('ascii')
        print('Predicted label is {}'.format(result))
```

5. To see if the model is making accurate predictions, check the output from this step against the numbers you plotted in the previous step.
You have now trained, deployed, and validated your first model in Amazon SageMaker.

**Next Step**

**Step 9: Clean Up (p. 43)**

**Step 7.2: Validate a Model Deployed with Batch Transform**

You now have a file in Amazon S3 that contains inferences that you got by running a batch transform job in **Step 6.2: Deploy the Model with Batch Transform (p. 37)**. To validate the model, check a subset of the inferences from the file to see whether they match the actual numbers from the test dataset.

**To validate the batch transform inferences**

1. Download the test data from Amazon S3.

   ```
s3 = boto3.resource('s3')
test_key = "/{}/test/examples".format(prefix)
s3.Bucket(bucket).download_file(test_key, 'test_data')
   ```

2. Plot the first 10 images from the test dataset with their labels.

   ```
   %matplotlib inline
   for i in range (0, 10):
     img = test_set[0][i]
     label = test_set[1][i]
     img_reshape = img.reshape((28,28))
     imgplot = plt.imshow(img_reshape, cmap='gray')
     print('This is a {}'.format(label))
     plt.show()
   ```

3. Download the output from the batch transform job from Amazon S3 to a local file.

   ```
s3.Bucket(bucket).download_file(prefix + '/batch-inference/examples.out', 'batch_results')
   ```

4. Get the first 10 results from the batch transform job.
with open('batch_results') as f:
    results = f.readlines()
for j in range (0, 10):
    print(results[j])

5. To see if the batch transform job made accurate predictions, check the output from this step against the numbers that you plotted from the test data.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications (p. 43)

Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications

In a production environment, you might have an internet-facing application sending requests to the endpoint for inference. The following high-level example shows how to integrate your model endpoint into your application.

1. Create an IAM role that the AWS Lambda service principal can assume. Give the role permissions to call the Amazon SageMaker InvokeEndpoint API.
2. Create a Lambda function that calls the Amazon SageMaker InvokeEndpoint API.
3. Call the Lambda function from a mobile application. For an example of how to call a Lambda function from a mobile application using Amazon Cognito for credentials, see Tutorial: Using AWS Lambda as Mobile Application Backend.

Next Step

Step 9: Clean Up (p. 43)

Step 9: Clean Up

To avoid incurring unnecessary charges, use the AWS Management Console to delete the resources that you created for this exercise.

Note
If you plan to explore other exercises in this guide, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/ and delete the following resources:
   - The endpoint. Deleting the endpoint also deletes the ML compute instance or instances that support it.
   - The endpoint configuration.
   - The model.
   - The notebook instance. Before deleting the notebook instance, stop it.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/ and delete the bucket that you created for storing model artifacts and the training dataset.
3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/ and delete all of the log groups that have names starting with /aws/sagemaker/.
Use Amazon SageMaker Autopilot to Automate Model Development

Amazon SageMaker Autopilot simplifies the machine learning experience by helping you explore your data and try different algorithms. It also automatically trains and tunes models on your behalf, to help you find the best algorithm. You simply upload tabular data in a file with comma-separated values (for example, from a spreadsheet or database), choose the target column to predict, and Autopilot builds a predictive model for you. These predictions can take the form of ordered numerical values (i.e., this is a regression model) or the form of categories (i.e., a classification model). Autopilot explores different combinations of data preprocessors, algorithms, and algorithm parameter settings to find an accurate model, similar to how a data scientist would, making it easier for novices to get started.

Topics
- Get Started (p. 45)
- Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio (p. 45)
- SageMaker Autopilot Notebook Output (p. 46)
- SageMaker Autopilot Problem Types (p. 47)

Get Started

Autopilot works with tabular data only. Autopilot is available in Amazon SageMaker Studio in the Create Experiment tab. It is recommended that you familiarize yourself with the features, and then try it out with your own data. The general flow is as follows:

1. Launch a new experiment, specify the location of the input data and where you want the outputs stored. The outputs include intermediate preprocessed data, the candidate generation notebook, the data exploration notebook, and the model artifacts.
2. Choose the column in your dataset that you want Autopilot to predict.
3. (Optional) Specify additional parameters, such as problem type, security configuration, or job completion criteria.
4. Launch Autopilot.

In addition to the visual experience in SageMaker Studio, there is an example notebook for Autopilot that can help you understand the flow using the AWS SDK.

Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio

1. Open Amazon SageMaker Studio and login.
2. Choose the Experiments tab (it looks like a conical flask).
3. Choose **Create experiment**.
4. Enter the experiment's details in the **Job Settings** form.
   - Name of the experiment - Must be unique to this experiment in the current AWS Region.
   - Input dataset S3 location - An s3:// formatted URL where SageMaker has read permissions.
   - Target attribute - This is the column of your data you want the model to target.
   - Output S3 location - An s3:// formatted URL where SageMaker has write permissions.
   - (Optional) Tell Autopilot whether you want a regression (e.g., house prices), binary classification (e.g., hotdog or not hotdog), or multi-class classification (e.g., cat, dog or bird) model. If you don’t specify this, Autopilot infers it from the values of the attribute you want to predict. In some cases, Autopilot is unable to infer accurately, in which case you must provide the value for the job to succeed.
   - (Optional) Tell Autopilot the metric you want it to use to evaluate models. If you don’t specify a metric, Autopilot makes the decision on your behalf based on the best metrics for the situation.
   - (Optional) Configure other job parameters. For example, security configuration or job completion criteria. In the latter case, if you are looking to limit the total wall clock time of the AutoML job, you can specify to limit the number of seconds/minutes the job is allowed to run.
   - (Optional) Specify **Generate Candidate Definitions Only**. In this case, instead of executing the entire AutoML workflow on autopilot, Autopilot stops execution after generating the notebooks for data exploration and candidate generation. This way, you can use the notebooks as a starting point to guide your own process of data exploration and model training/tuning. Both notebooks have highlighted sections that explain what kinds of changes are typical, such as changing instance type, cluster size, and so on.
5. Choose to run the training immediately, or only produce data exploration and candidate generation notebooks.

This is all you need to run a Autopilot experiment. The process will generate a model as well as statistics that you can view in real time while the experiment is running. After the experiment is completed, you can view the trials, sort by objective metric, and right-click to deploy the model for use in other environments.

**SageMaker Autopilot Notebook Output**

During the analysis phase of the AutoML job, two notebooks are created that describe the plan that Autopilot will follow to generate candidate models. First, there’s a data exploration notebook, that describes what Autopilot learned about the data that you provided. Second, there’s a candidate generation notebook, which uses the information about the data to generate candidates. Together, these notebooks describe the plan that would be executed on autopilot if you don’t choose to stop with candidate generation only.

You can run both notebooks in SageMaker or locally if you have installed the SageMaker Python SDK. You can share the notebooks just like any other SageMaker Studio notebook. Modifications on the candidate generation notebook is encouraged. The notebooks are created for you to experiment. When you run the notebooks in your default instance you will incur baseline costs, but when you execute HPO jobs from the candidate notebook, these jobs use additional compute resources that will incur additional costs.

**Candidate Generation Notebook**

The candidate generation notebook contains each suggested preprocessing step, suggested algorithm, and suggested hyperparameter ranges. If you chose to only produce the notebook and not run the AutoML job, you can then decide which candidates to be trained and tuned. They optimize automatically
and a final, best candidate will be identified. If you ran the job directly without seeing the candidates first, the best candidate is displayed when you open the notebook after the job’s completion.

**Data Exploration Notebook**

A second notebook is produced during the analysis phase of the AutoML job that helps you identify problems in your dataset. It identifies specific areas for investigation in order to help you identify upstream problems with your data that may result in a suboptimal model.

**SageMaker Autopilot Problem Types**

You have the option of focusing the Autopilot experiment on specific problem types, which in turn limits the kind of preprocessing and algorithms that are tried.

Your problem type options are as follows:

**Topics**
- Linear regression (p. 47)
- Binary classification (p. 47)
- Multi-class classification (p. 47)
- Automatic problem type detection (p. 48)

Each of the problem types require a tabular data input with the columns labelled. In a typical ML environment you set the feature that you are interested in predicting (objective) and you can also specify the objective type (classification or regression). However, with Autopilot these are optional. It can detect these settings for you.

**Linear regression**

One commonly used linear regression example is home prices prediction. Provided a dataset of home prices and other features like number of bathrooms and bedrooms, linear regression can create a model for you that takes one or more of these features as an input and then predicts a home price. When using your own data, it is possible to provide some missing observations (blank data), but you are notified when there's too much missing data for a good predictive model.

**Binary classification**

Binary classification models are trained using labeled examples of objects mixed with other examples that are not that object. For example, there is the Not Hotdog app whose primary function was to evaluate images and tell you if it contains a hotdog or not. While this provides a humorous look at an AI application, a real-world example is evaluating the Titanic dataset and making fatality predictions (alive or dead). Being alive or dead is a binary outcome that can be easily measured. The provided features such as cabin class, life boat number, or age could all be considered reasonable survivability factors for an ill-fated transatlantic sea voyage.

**Multi-class classification**

Multi-class refers to the range of possible predictions the model will make. For example, an emotion detection model might have a handful of possible classes: happy, sad, angry, or surprised. A model of this type would not only predict each class, it would return the percentage probability of each class. Another example is found in self-driving cars where they use models to identify objects like pedestrians, vehicles, stop signs, and green/yellow/red lights.
Automatic problem type detection

When setting a problem type with the AutoML API, you have the option of defining one, or letting Autopilot detect it on your behalf. When the job is finished, if you set a ProblemType, the ResolvedAttribute's ProblemType will match the ProblemType you set. If you left it blank (or null), the ProblemType will be whatever Autopilot decides on your behalf.
Prepare and Label Data

Prepare and label data in Amazon SageMaker.

Topics
- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- Using Amazon Augmented AI for Human Review (p. 105)
- Create and Manage Workforces (p. 131)
- HTML Elements Reference (p. 142)

Use Amazon SageMaker Ground Truth for Labeling

To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models. With Ground Truth, you can use workers from either Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce along with machine learning to enable you to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

In order to automate labeling your training dataset, you can optionally use automated data labeling, a Ground Truth process that uses machine learning to decide which data needs to be labeled by humans. Automated data labeling may reduce the labeling time and manual effort required. For more information, see Using Automated Data Labeling (p. 65).

Use either pre-built or custom tools to assign the labeling tasks for your training dataset. A labeling UI template is a webpage that Ground Truth uses to present tasks and instructions to your workers. The Amazon SageMaker console provides built-in templates for labeling data. You can use these templates to get started, or you can build your own tasks and instructions by using our HTML 2.0 components. For more information, see Creating Custom Labeling Workflows (p. 84).

Use the workforce of your choice to label your dataset. You can choose your workforce from:
- The Amazon Mechanical Turk workforce of over 500,000 independent contractors worldwide.
- A private workforce that you create from your employees or contractors for handling data within your organization.
- A vendor company that you can find in the AWS Marketplace that specializes in data labeling services.

For more information, see Create and Manage Workforces (p. 131).

You store your datasets in Amazon S3 buckets. The buckets contain three things: The data to be labeled, an input manifest file that Ground Truth uses to read the data files, and an output manifest file. The output file contains the results of the labeling job. For more information, see Use Input and Output Data (p. 71).

Events from your labeling jobs appear in Amazon CloudWatch under the /aws/sagemaker/LabelingJobs group. CloudWatch uses the labeling job name as the name for the log stream.
Are You a First-time User of Ground Truth?

If you are a first-time user of Ground Truth, we recommend that you do the following:

1. **Read Getting started (p. 50)**—This section walks you through setting up your first Ground Truth labeling job.

2. **Explore other topics**—Depending on your needs, do the following:
   - **Create instruction pages for your labeling jobs**—Create a custom instruction page that makes it easier for your workers to understand the requirements of the job. For more information, see Creating Instruction Pages (p. 82).
   - **Manage your labeling workforce**—Create new work teams and manage your existing workforce. For more information, see Create and Manage Workforces (p. 131).
   - **Create a custom UI**—Make it easier for your workers to quickly and correctly label your data by creating a custom UI for them to use. For more information, see Creating Custom Labeling Workflows (p. 84).

3. **See the API Reference (p. 843)**—This section describes operations to automate Ground Truth operations.

---

**Getting started**

To get started using Amazon SageMaker Ground Truth, follow the instructions in the following sections. The sections here explain how to use the console to create a labeling job, assign a public or private workforce, and send the labeling job to your workforce. You can also learn how to monitor the progress of a labeling job.

If you want to create a custom labeling job, see Creating Custom Labeling Workflows (p. 84) for instructions.

Before you create a labeling job, you must upload your dataset to an Amazon S3 bucket. For more information, see Use Input and Output Data (p. 71).

Topics

- Step 1: Before You Begin (p. 50)
- Step 2: Create a Labeling Job (p. 51)
- Step 3: Select Workers (p. 52)
- Step 4: Configure the Bounding Box Tool (p. 53)
- Step 5: Monitoring Your Labeling Job (p. 53)

**Step 1: Before You Begin**

Before you begin using the Amazon SageMaker console to create a labeling job, you must set up the dataset for use. Do this:

1. Save two images at publicly available HTTP URLs. The images are used when creating instructions for completing a labeling task. The images should have an aspect ratio of around 2:1. For this exercise, the content of the images is not important.
2. Create an Amazon S3 bucket to hold the input and output files. The bucket must be in the same Region where you are running Ground Truth. Make a note of the bucket name because you use it during step 2.
3. Place 5–10 PNG images in the bucket.
4. Create a manifest file for the dataset and store it in the S3 bucket. Use these steps:
Getting started


b. Add a line similar to the following for each image file in your dataset:

```json
{s3://bucket/path/imageFile.png}
```

Add one line for each PNG file in your S3 bucket.

c. Save the file in the S3 bucket containing your source files. Record the name because you use it in step 2.

**Note**

It is not necessary to store the manifest file in the same bucket as the source file. You use the same bucket in this exercise because it is easier.

For more information, see Input Data (p. 71).

Assign the following permissions policy to the user that is creating the labeling job:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "sagemakergroundtruth",
      "Effect": "Allow",
      "Action": [
        "cognito-idp:CreateGroup",
        "cognito-idp:CreateUserPool",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:CreateUserPoolClient",
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:UpdateUserPool"
      ],
      "Resource": "*
    }
  ]
}
```

Next

Step 2: Create a Labeling Job (p. 51)

**Step 2: Create a Labeling Job**

In this step you use the console to create a labeling job. You tell Amazon SageMaker Ground Truth the Amazon S3 bucket where the manifest file is stored and configure the parameters for the job. For more information about storing data in an Amazon S3 bucket, see Use Input and Output Data (p. 71).

**To create a labeling job**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. From the left navigation, choose **Labeling jobs**.
3. Choose **Create labeling job** to start the job creation process.
4. In the **Job overview** section, provide the following information:
• **Job name** — Give the labeling job a name that describes the job. This name is shown in your job list. The name must be unique in your account in an AWS Region.

• **Label attribute name** — Leave this unchecked as the default value is the best option for this introductory job.

• **Input dataset location** — Enter the S3 location of the manifest file that you created in step 1.

• **Output dataset location** — the location where your output data is written.

• **IAM role** — Create or choose an IAM role with the SageMakerFullAccess IAM policy attached.

5. In the **Task type** section, for the **Dataset type** field, choose **Bounding box** as the task type.

6. Choose **Next** to move on to configuring your labeling job.

**Next**

**Step 3: Select Workers (p. 52)**

**Step 3: Select Workers**

In this step you choose a workforce for labeling your dataset. You can create your own private workforce or you can use the Amazon Mechanical Turk workforce. If you create a private workforce in this step you won't be able to import your Amazon Cognito user pool later. For more information, see Manage a Private Workforce (p. 136). Use the Amazon Mechanical Turk workforce for this exercise instead.

You can create a private workforce to test Amazon SageMaker Ground Truth. Use email addresses to invite the members of your workforce.

**To create a private workforce**

1. In the **Workers** section, choose **Private**.

2. If this is your first time using a private workforce, in the **Email addresses** field, enter up to 100 email addresses. The addresses must be separated by a comma. You should include your own email address so that you are part of the workforce and can see data object labeling tasks.

3. In the **Organization name** field, enter the name of your organization. This information is used to customize the email sent to invite a person to your private workforce.

4. In the **Contact email** field enter an email address that members of the workforce use to report problems with the task.

If you choose to use the Amazon Mechanical Turk workforce to label the dataset, you are charged for labeling tasks completed on the dataset.

**To use the Amazon Mechanical Turk workforce**

1. In the **Workers** section, choose **Public**.

2. Choose **The dataset does not contain PII** to acknowledge that the dataset does not contain any personally identifiable information.

3. Choose **The dataset does not contain adult content**. to acknowledge that the sample dataset has no adult content.

4. Review and accept the statement that the dataset will be viewed by the public workforce.

**Next**

**Step 4: Configure the Bounding Box Tool. (p. 53)**
Step 4: Configure the Bounding Box Tool.

Finally you configure the bounding box tool to give instructions to your workers. You can configure a task title that describes the task and provides high-level instructions for the workers. You can provide both quick instructions and full instructions. Quick instructions are displayed next to the image to be labeled. Full instructions contain detailed instructions for completing the task. In this example, you only provide quick instructions. You can see an example of full instructions by choosing Full instructions at the bottom of the section.

To configure the bounding box tool

1. In the Task description field type in brief instructions for the task. For example:

   Draw a box around any objects in the image.

   Replace objects with the name of an object that appears in your images.

2. In the Labels field, type a category name for the objects that the worker should draw a bounding box around. For example, if you are asking the worker to draw boxes around football players, you could use "FootballPlayer" in this field.

3. The Short instructions section enables you to create instructions that are displayed on the page with the image that your workers are labeling. We suggest that you include an example of a correctly drawn bounding box and an example of an incorrectly drawn box. To create your own instructions, use these steps:
   a. Select the text between GOOD EXAMPLE and the image placeholder. Replace it with the following text:

      Draw the box around the object with a small border.

   b. Select the first image placeholder and delete it.
   c. Choose the image button and then enter the HTTPS URL of one of the images that you created in step 1.
   d. Select the text between BAD EXAMPLE and the image placeholder. Replace it with the following text:

      Don't make the bounding box too large or cut into the object.

   e. Select the second image placeholder and delete it.
   f. Choose the image button and then enter the HTTPS URL of the other image that you created in step 1.

Configuration of your labeling job is complete. To start your job, choose Submit.

Next

Step 5: Monitoring Your Labeling Job (p. 53)

Step 5: Monitoring Your Labeling Job

After you create your labeling job, you see a list of all the jobs that you have created. You can use this list to monitor that status of your labeling jobs. The list has the following fields:

- **Name**—The name that you assigned the job when you created it.
- **Status**—The completion status of the job. The status can be one of Complete, Failed, In progress, or Stopped.
- **Labeled objects/total**—Shows the total number of objects in the labeling job and how many of them have been labeled.
• **Creation time**—The date and time that you created the job.

You can also clone, chain, or stop a job. Select a job and then select one of the following from the **Actions** menu:

• **Clone**—Creates a new labeling job with the configuration copied from the selected job. You can clone a job when you want to change to the job and run it again. For example, you can clone a job that was sent to a private workforce so that you can send it to the Amazon Mechanical Turk workforce. Or you can clone a job to rerun it against a new dataset stored in the same location as the original job.

• **Chain**—Creates a new labeling job that can build upon the data and models (if any) of a stopped, failed, or completed job. For more information about the use cases and how to use it, see [Chaining labeling jobs](p. 66).

• **Stop**—Stops a running job. You cannot restart a stopped job. You can clone a job to start over or chain the job to continue from where it left off. Labels for any already labeled objects are written to the output file location. For more information, see [Output Data](p. 75).

**Task Type Templates**

Amazon SageMaker Ground Truth provides templates for image- and text-based labeling jobs. The following topics describe each template task type. To learn how to create a labeling job in the console using one of these templates, see [Getting started (p. 50)](Getting started). To learn how to create a labeling job using the Ground Truth API, see [CreateLabelingJob](CreateLabelingJob). To learn how to create a custom labeling workflow, see [Creating Custom Labeling Workflows (p. 84)](Creating Custom Labeling Workflows).

**Topics**

• Bounding Box (p. 54)
• Image Classification (p. 55)
• Semantic Segmentation (p. 56)
• Label Verification (p. 58)
• Text Classification (p. 59)
• Named Entity Recognition (p. 59)

**Bounding Box**

In many situations, the images used to train a machine learning model contain more than one object. Use the Amazon SageMaker Ground Truth bounding box labeling job type to classify and localize multiple objects within images. In Ground Truth, you can create single- and multi-class bounding box labeling jobs. When you choose the bounding box task type, your workers are asked to identify one or more objects in your images by drawing a bounding box around the object. Each bounding box is associated with a label.

For example, when performing a bounding box labeling job task, workers might see instructions and labels similar to the following.
You can create a bounding box labeling job using the Amazon SageMaker console or API. To learn how to start a bounding box labeling job in the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.

**Image Classification**

Use an Amazon SageMaker Ground Truth image classification labeling task when you need workers to classify images using predefined labels that you specify. Workers are shown images and are asked to choose one label for each image.

For example, workers might see instructions and labels similar to the following.
You can create an image classification labeling job using the Amazon SageMaker console or API. To learn how to start an image classification labeling job using on the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.

**Semantic Segmentation**

To identify the contents of an image at the pixel level, use an Amazon SageMaker Ground Truth semantic segmentation labeling task. When assigned a semantic segmentation labeling job, workers classify pixels in the image into a set of predefined labels or classes. Ground Truth supports single and multi-class semantic segmentation labeling jobs.

For example, when assigned a semantic segmentation task in Ground Truth, workers might see instructions and labels similar to the following.
Instructions

View full instructions

View tool guide

How to use the Auto-segment tool

Good example

All pixels in the image that are part of an animal have been colors with the appropriate label color.

Bad example

Some animals in the image have not been colored in completely.

The color for a given animal extends beyond the boundaries of the animal.
Images that contain large numbers of objects that need to be segmented require more time. To help workers label these objects in less time and with greater accuracy, Ground Truth provides an AI-assisted auto-segmentation tool. For information, see Auto-Segmentation Tool (p. 60).

**Note**
The auto-segmentation tool is available in all segmentation tasks that are sent to a private workforce or vendor workforce. It isn’t available for tasks sent to the public workforce (Amazon Mechanical Turk).

You can create a semantic segmentation labeling job using the Amazon SageMaker console or API. To learn how to start a semantic segmentation labeling job on the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.

## Label Verification

Building a highly accurate training dataset for your machine learning (ML) algorithm is an iterative process. Typically, you review and continuously adjust your labels until you are satisfied that they accurately represent the ground truth, or what is directly observable in the real world.

You can use an Amazon SageMaker Ground Truth label verification task to direct workers to review a dataset’s labels and improve label accuracy. Workers can indicate if the existing labels are correct or rate label quality. They can also add comments to explain their reasoning. Amazon SageMaker Ground Truth supports label verification for Bounding Box (p. 54) and Semantic Segmentation (p. 56) labels.

For example, when performing a label verification task, workers might see instructions and labels similar to the following.

You can create a label verification labeling job using the Amazon SageMaker console or API. To learn how to start a label verification job on the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.
Text Classification

To categorize articles and text into predefined categories, use text classification. For example, you can use text classification to identify the sentiment conveyed in a review or the emotion underlying a section of text. Use Amazon SageMaker Ground Truth text classification to have workers sort text into categories that you define.

For example, when performing a text classification job task, workers might see instructions and labels similar to the following.

You can create a text classification labeling job using the Amazon SageMaker console or API. To learn how to start a text classification labeling job on the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.

Named Entity Recognition

To extract information from unstructured text and classify it into predefined categories, use an Amazon SageMaker Ground Truth named entity recognition (NER) labeling task. Traditionally, NER involves sifting through text data to locate noun phrases, called named entities, and categorizing each with a label, such as “person,” “organization,” or “brand.” You can broaden this task to label longer spans of text and categorize those sequences with predefined labels that you specify.

When tasked with a named entity recognition labeling job, workers apply your labels to specific words or phrases within a larger text block. They choose a label, then apply it by using the cursor to highlight the part of the text to which the label applies. Workers can’t apply multiple labels to the same text, and labels can’t overlap.

When performing a named entity recognition task in Ground Truth, workers might see instructions and labels similar to the following.
You can create a named entity recognition labeling job using the Amazon SageMaker console or API. To learn how to start a named entity recognition labeling job using on the console, see Getting started (p. 50). To use the API, see CreateLabelingJob.

Auto-Segmentation Tool

Image segmentation is the process of dividing an image into multiple segments, or sets of labeled pixels. In Amazon SageMaker Ground Truth, the process of identifying all pixels that fall under a given label involves applying a colored filler, or "mask", over those pixels. Some labeling job tasks contain images with a large numbers of objects that need to be segmented. To help workers label these objects in less time and with greater accuracy, Ground Truth provides an auto-segmentation tool for segmentation tasks assigned to private and vendor workforces. This tool uses a machine learning model to automatically segment individual objects in the image with minimal worker input. Workers can refine the mask generated by the auto-segmentation tool using other tools found in the worker console. This helps workers complete image segmentation tasks faster and more accurately, resulting in lower cost and higher label quality.

Note
The auto-segmentation tool is available for segmentation tasks that are sent to a private workforce or vendor workforce. It isn't available for tasks sent to the public workforce (Amazon Mechanical Turk).

Tool Preview

When workers are assigned a labeling job that provides the auto-segmentation tool, they are provided with detailed instructions on how to use the tool. For example, a worker might see the following in the worker console:
Instructions

View full instructions

View tool guide

How to use the Auto-segment tool

Good example

All pixels in the image that are part of an animal have been colors with the appropriate label color.

Bad example

Some animals in the image have not been colored in completely.

The color for a given animal extends beyond the boundaries of the animal.
Workers can use **View full instructions** to learn how to use the tool. Workers will need to place a point on four extreme-points (top-most, bottom-most, left-most, and right-most points) of the object of interest, and the tool will automatically generate a mask for the object. Workers can further-refine the mask using the other tools provided, or by using the auto-segmentation tool on smaller portions of the object that were missed.

**Tool Availability**

The auto-segmentation tool automatically appears in your workers’ consoles if you create a semantic segmentation labeling job using the Amazon SageMaker console. While creating a semantic segmentation job in the Amazon SageMaker console, you will be able to preview the tool while creating worker instructions. To learn how to create a semantic segmentation labeling job in the Amazon SageMaker console, see Getting started (p. 50).

If you are creating a custom instance segmentation labeling job in the Amazon SageMaker console or creating an instance- or semantic-segmentation labeling job using the Ground Truth API, you need to create a custom task template to design your worker console and instructions. To include the auto-segmentation tool in your worker console, ensure that the following conditions are met in your custom task template:

- For semantic segmentation labeling jobs created using the API, the `<crowd-semantic-segmentation>` is present in the task template. For custom instance segmentation labeling jobs, the `<crowd-instance-segmentation>` tag is present in the task template.
- The task is assigned to a private workforce or vendor workforce.
- The images to be labeled are Amazon Simple Storage Service Amazon S3) objects that have been pre-signed for the Worker so that they can access it. This is true if the task template includes the `grant_read_access` filter. For information about the `grant_read_access` filter, see Adding automation with Liquid (p. 88).

The following is an example of a custom task template for a custom instance segmentation labeling job, which includes the `<crowd-instance-segmentation/>` tag and the `grant_read_access` Liquid filter.

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-instance-segmentation
    name="crowd-instance-segmentation"
    src="{{ task.input.taskObject | grant_read_access }}"
    labels="['Car','Road']">
    <full-instructions header="Segmentation instructions">
      Segment each instance of each class of objects in the image.
    </full-instructions>
    <short-instructions>
      <p>Segment each instance of each class of objects in the image.</p>
      <h3 style="color: green">GOOD EXAMPLES</h3>
      <img src="path/to/image.jpg" style="width: 100%"
      <p>Good because A, B, C.</p>
      <h3 style="color: red">BAD EXAMPLES</h3>
      <img src="path/to/image.jpg" style="width: 100%"
      <p>Bad because X, Y, Z.</p>
    </short-instructions>
  </crowd-instance-segmentation>
</crowd-form>
```
Data Labeling

Amazon SageMaker Ground Truth manages sending your data objects to workers to be labeled. Labeling each data object is a task. Workers complete each task until the entire labeling job is complete. Ground Truth divides the total number of tasks into smaller batches that are sent to workers. A new batch is sent to workers when the previous one is finished.

Ground Truth provides two features that help improve the accuracy of your data labels and reduce the total cost of labeling your data.

The first feature is annotation consolidation. This helps to improve the accuracy of your data object’s labels. It combines the results of multiple worker’s annotation tasks into one high-fidelity label.

The second feature is automated data labeling. This uses machine learning to label portions of your data automatically without having to send them to human workers.

Topics
- Batches for Labeling Tasks (p. 63)
- Annotation Consolidation (p. 63)
- Using Automated Data Labeling (p. 65)
- Chaining labeling jobs (p. 66)
- Label verification and adjustment (p. 69)

Batches for Labeling Tasks

Amazon SageMaker Ground Truth sends data objects to your workers in batches. There are one or more tasks for each data object. For each task, a worker annotates one of your data objects. A batch provides the following:

- It sets the number of data objects that are available to workers. After the objects are annotated another batch is sent.
- It breaks the work into smaller chunks to avoid overloading your workforce.
- It provides chunks of data for the iterative training of automated labeling models.

Ground Truth first sends a batch of 10 tasks to your workers. It uses this small batch to set up the labeling job and to make sure that the job is correctly configured.

After the small batch, Ground Truth sends larger batches to your workers. You can configure the batch size when you create the job using the CreateLabelingJob (p. 897). When you use the Amazon SageMaker console, your job uses 1,000 tasks in each batch.

Annotation Consolidation

Annotation consolidation combines the annotations of two or more workers into a single label for your data objects. An annotation is the result of a single worker. Annotation consolidation combines multiple annotations from different workers to come up with a probabilistic estimate of what the true label should be. The label is assigned to each object in the dataset. Each object in the dataset typically has multiple annotations but only one label or set of labels.

You can decide how many workers should annotate each object in your dataset. More workers can increase the accuracy of your labels but also increases the cost of labeling. Amazon SageMaker Ground Truth uses the following defaults in the Amazon SageMaker console. When you use the
CreateLabelingJob (p. 897) operation, you set the number of workers that should annotate each data 
object using the NumberOfHumanWorkersPerDataObject parameter.

- **Text classification**—3 workers
- **Image classification**—3 workers
- **Bounding boxes**—5 workers
- **Semantic segmentation**—3 workers
- **Named entity recognition**—3 workers

You can override the default number of workers that label a data object using the console or the 
CreateLabelingJob (p. 897) operation.

Ground Truth provides an annotation consolidation function for each of its predefined labeling tasks: 
Bounding box, image classification, semantic segmentation, and text classification. These are the 
functions:

- Multi-class annotation consolidation for image and text classification uses a variant of the Expectation 
  Maximization approach to annotations. It estimates parameters for each worker and uses Bayesian 
  inference to estimate the true class based on the class annotations from individual workers.
- Bounding box annotation consolidates bounding boxes from multiple workers. This function finds the 
  most similar boxes from different workers based on the Jaccard index, or intersection over union, of 
  the boxes and averages them.
- Semantic segmentation annotation consolidation treats each pixel in a single image as a multi-
  class classification. This function treats the pixel annotations from workers as “votes,” with more 
  information from surrounding pixels incorporated by applying a smoothing function to the image.
- Named entity recognition clusters text selections by Jaccard similarity and calculates selection 
  boundaries based on the mode, or median if the mode isn’t clear. The label resolves to the most 
  assigned entity label in the cluster, breaking ties by random selection.

**Note**
If you want to run worker responses through different algorithms on your own, that data is 
stored in the [project-name]/annotations/worker-response folder of the Amazon S3 
bucket where you direct the job output.

Creating Your Own Annotation Consolidation Function

There are many possible approaches for writing an annotation consolidation function. The approach 
that you take depends on the nature of the annotations to consolidate. Broadly, consolidation functions 
look at the annotations from workers, measure the similarity between them, and then use some form of 
probabilistic judgment to determine what the most probable label should be.

Assessing Similarity

To assess the similarity between labels, you can use one of the following strategies or you can use one 
that meets your data labeling needs:

- For label spaces that consist of discrete, mutually exclusive categories, such as multi-class 
  classification, assessing similarity can be straightforward. Discrete labels either match or not.
- For label spaces that don't have discrete values, such as bounding box annotations, find a broad 
  measure of similarity. For bounding boxes, one such measure is the Jaccard index. This measures 
  the ratio of the intersection of two boxes with the union of the boxes to assess how similar they 
  are. For example, if there are three annotations, then there can be a function that determines which 
  annotations represent the same object and should be consolidated.
Assessing the Most Probable Label

With one of the above strategies in mind, make some sort of probabilistic judgment on what the consolidated label should be. In the case of discrete, mutually exclusive categories this can be straightforward. One of the most common ways to do this is to take the results of a majority vote between the annotations. This weights the annotations equally.

Some approaches attempt to estimate the accuracy of different annotators and weight their annotations in proportion to the probability of correctness. An example of this is the Expectation Maximization method, which is used in the default Ground Truth consolidation function for multi-class annotations.

For more information about creating an annotation consolidation function, see Step 3: Processing with AWS Lambda (p. 102).

Using Automated Data Labeling

Ground Truth can use active learning to automate the labeling of your input data. Active learning is a machine learning technique that identifies data that should be labeled by your workers.

Automated data labeling is optional. Turn it on when you create a labeling job. Automated data labeling incurs Amazon SageMaker training and inference costs, but it helps to reduce the cost and time that it takes to label your dataset compared to humans alone.

Use automated data labeling on large datasets. The neural networks used with active learning require a significant amount of data for every new dataset. With larger datasets, there is more potential to automatically label the data and therefore reduce the total cost of labeling. We recommend that you use thousands of data objects when using automated data labeling. The system minimum for automated labeling is 1,250 objects, but to get a meaningful amount of your data automatically labeled, we strongly suggest a minimum with 5,000 or more objects.

The potential benefit of automated data labeling also depends on the accuracy that you require. Higher accuracy levels generally reduce the number of data objects that are automatically labeled.

When Amazon SageMaker Ground Truth starts an automated data labeling job, it first selects a random sample of the input data. Then it sends the sample to human workers. When the labeled data are returned, Ground Truth uses this set of data as validation data. It is used to validate the machine learning models that Ground Truth trains for automated data labeling.

Next, Ground Truth runs an Amazon SageMaker batch transform using the validation set. This generates a quality metric that Ground Truth uses to estimate the potential quality of auto-labeling the rest of the unlabeled data.

Ground Truth next runs an Amazon SageMaker batch transform on the unlabeled data in the dataset. Any data where the expected quality of automatically labeling the data is above the requested level of accuracy is considered labeled.

After performing the auto-labeling step, Ground Truth selects a new sample of the most ambiguous unlabeled data points in the dataset. It sends those to human workers. Ground Truth uses the existing labeled data and this additional labeled data from human workers to train a new model. The process is repeated until the dataset is fully labeled.

Note
Input data quotas apply for automated labeling jobs. See Input Data Quotas (p. 72) for information about dataset size, input data size and resolution limits for automated Semantic Segmentation, Object Detection, and Image Classification.

Ensure the automated-labeling model is ready for production use
The model generated by your labeling job needs fine-tuning and/or testing before you use it in production. Fine-tune the model generated by Ground Truth (or create and tune another
supervised model of your choice) on the dataset produced by your labeling job. Optimize the model’s architecture and hyperparameters. If you decide to use the model for inference without fine-tuning, we strongly recommend making sure its accuracy is evaluated on a representative (e.g. randomly selected) subset of the dataset labeled with Ground Truth and matches your expectations.

**Amazon EC2 Instances Required for Automated Data Labeling**

To run automated data labeling, Ground Truth requires the following Amazon EC2 resources for training and batch inference jobs:

<table>
<thead>
<tr>
<th>Automated labeling action</th>
<th>Training instance type</th>
<th>Inference instance type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>ml.p3.2xlarge*</td>
<td>ml.c5.xlarge</td>
</tr>
<tr>
<td>Object detection</td>
<td>ml.p3.2xlarge*</td>
<td>ml.c5.4xlarge</td>
</tr>
<tr>
<td>Text classification</td>
<td>ml.c5.2xlarge</td>
<td>ml.m4.xlarge</td>
</tr>
<tr>
<td>Semantic Segmentation</td>
<td>ml.p3.2xlarge*</td>
<td>ml.p3.2xlarge*</td>
</tr>
</tbody>
</table>

*ml.p2.8xlarge is substituted in the following regions: Mumbai (ap-south-1)*

**A note about pricing**

Automated labeling incurs two separate charges: the per item charge (Ground Truth pricing), and the charge for the Amazon EC2 instance required to run the model (Amazon EC2 pricing).

These instances are managed by Ground Truth. They are created, configured, and destroyed as needed to perform your job. They do not appear in your Amazon EC2 instance dashboard.

**Chaining labeling jobs**

Amazon SageMaker Ground Truth can reuse datasets from prior jobs in two ways: cloning and chaining.

Cloning is a relatively straightforward operation. Cloning copies the set-up of a prior labeling job and allows you to make additional changes, before setting it to run.

Chaining is a more complex operation. Chaining uses not only the setup of the prior job, but the results. This allows you to continue an incomplete job, and add labels or data objects to a completed job.

When it comes to the data being processed:

- **Cloning** — uses the prior job’s input manifest, with optional modifications, as the new job’s input manifest.
- **Chaining** — uses the prior job’s output manifest as the new job’s input manifest.

**Chaining labeling jobs**

Some situations where chaining is useful include:

- Continue a labeling job that was manually stopped.
- Continue a labeling job that failed mid-job, with issues fixed.
- Switch to automated labeling after manually labeling part of a job (or vice-versa).
- Add more data objects to a completed job and start the job from there.
• Add another annotation to a completed job. For example, you have a collection of phrases labeled for
topic, then want to run the set again, categorizing them by the topic’s implied audience.

In Amazon SageMaker Ground Truth you can configure a chained labeling job via either the console or API.

**Key Term: Label attribute name**

The label attribute name (LabelAttributeName in the API) is a string used as the key for the key-value pair formed with the label that a worker assigns to the data object.

There are a few rules for the label attribute name.

• It cannot end with `-metadata`.
• The names `source` and `source-ref` are reserved and cannot be used.
• Semantic-segmentation labeling jobs require it to end with `-ref`. All other labeling jobs require it to
not end with `-ref`. The adding of `-ref` is managed automatically for you in jobs configured via the console.
• If you’re using the same label attribute name from the originating job and you configure the chained job to use auto-labeling, then if it had been in auto-labeling mode at any point, the model from the originating job is used.

In an output manifest, it can appear something like this:

```json
"source-ref": "<S3 URI>",
"<Label attribute name>": {
  "annotations": [{
    "class_id": 0,
    "width": 99,
    "top": 87,
    "height": 62,
    "left": 175
  }],
  "image_size": [{
    "width": 344,
    "depth": 3,
    "height": 234
  }]
},
"<Label attribute name>-metadata": {
  "job-name": "<job name>",
  "class-map": {
    "0": "<Label attribute name>"
  },
  "human-annotated": "yes",
  "objects": [{
    "confidence": 0.09
  }],
  "creation-date": "<timestamp>",
  "type": "groundtruth/object-detection"
}
```

If you're creating a job in the console, the job name is used as the label attribute name for the job if you
don't explicitly set the label attribute name value.

**Starting a chained job in the console**

Select a stopped, failed, or completed labeling job from the list of your existing jobs. This enables the
**Actions** menu.
From the **Actions** menu, select **Chain**.

**Job overview panel**

In the **Job overview** panel, a new **Job name** is set based on the title of the job from which you are chaining this one. You can change it.

You may also specify a label attribute name different from the labeling job name.

If you’re chaining from a completed job, the label attribute name uses the name of the new job you’re configuring. To change the name, select the check box.

If you’re chaining from a stopped or failed job, the label attribute name uses the name of the job from which you’re chaining. It’s easy to see and edit the value because the name check box is checked.

**Attribute label naming considerations**

- **The default** uses the label attribute name Ground Truth has selected. All data objects without data connected to that label attribute name are labeled.

- **Using a label attribute name** not present in the manifest causes the job to process all the objects in the dataset.

The **input dataset location** in this case is automatically selected as the output manifest of the chained job. The input field is not available, so you cannot change it.

**Adding data objects to a labeling job**

You cannot specify an alternate manifest file. Manually edit the output manifest from the previous job to add new items before starting a chained job. The S3 URI helps you locate where you are storing the manifest in your S3 bucket. Download the manifest file from there, edit it locally on your computer, then upload the new version to replace it. Make sure you are not introducing errors during editing. We recommend you use JSON linter to check your JSON. Many popular text editors and IDEs have linter plugins available.

**Starting a chained job with the API**

The procedure is almost the same as setting up a new labeling job with CreateLabelingJob, except for two primary differences.

- **Manifest location**: Rather than use your original manifest from the prior job, the value for the ManifestS3Uri in the DataSource should point to the S3 URI of the output manifest from the prior labeling job.

- **Label attribute name**: Setting the correct LabelAttributeName value is important here. As pointed out, this is the key portion of a key-value pair where labeling data is the value. Sample use cases include:
  - **Adding new or more-specific labels to a completed job** — set a new label attribute name.
  - **Labeling the unlabeled items from a prior job** — use the label attribute name from the prior job.

**Using a partially labeled dataset**

You can get some chaining benefits if you use an augmented manifest that has already been partially labeled. Check the **Label attribute name** check box and set the name so that it matches the name in your manifest.

If you’re using the API, the instructions are the same as starting a chained job. However, be sure to upload your manifest to an S3 bucket and use it instead of using the output manifest from a prior job.

The **Label attribute name** value in the manifest has to conform to the naming considerations discussed above.
Label verification and adjustment

When the labels on a dataset need to be validated, Ground Truth provides functionality to have workers to verify that labels are correct or to adjust previous labels.

These types of jobs fall into two distinct categories:

- **Label verification** — workers indicate if the existing labels are correct, or rate quality, and can add comments to explain their reasoning.
- **Label adjustment** — workers adjust prior annotations to correct them.

Start a label verification job in the console

1. Start a new labeling job by chaining (p. 66) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
2. Choose the **Label verification** task type and continue to the next screen.
3. In the **Display existing labels** pane, the system will detect and populate the available label attribute names in your manifest. Select the label attribute name for the prior labeling job you want to verify.
4. Use the instructions areas of the tool designer to provide context about what the previous labelers were asked to do and what the current verifiers need to check.
5. Use the **See preview** option to check that the tool is displaying the prior labels correctly and presents the label verification task clearly.

Start an adjustment job in the console

1. Start a new labeling job by chaining (p. 66) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
2. Choose the correct task type for your data and continue to the next screen.
3. After selecting the workers, there is an optional configuration section to **Display existing labels**. If it is not expanded, click the arrow next to the title to expand it.
4. Check the box next to **I want to display existing labels from the dataset for this job**.
5. Select the **Label attribute name** from your manifest that corresponds to the labels you want to display for adjustment. The system will try to detect and populate these values by analyzing the manifest, but you may need to set the correct value.
6. Use the instructions areas of the tool designer to provide context about what the previous labelers were tasked with doing and what the current verifiers need to check and adjust.
7. Use the **See preview** option to check that the tool is displaying the prior labels correctly and presents the task clearly.

Label verification and adjustment data in the output manifest

Label verification data is written to the **output manifest** within the metadata for the label. Two properties are added to the metadata:

- A **type** property with a value of "groundtruth/label-verification.
- A **worker-feedback** property with an array of **comment** values. This is only added when the worker enters comments. If there are no comments, the field will not appear.

```json
{
  "source-ref":"S3 bucket location",
  "verify-bounding-box":1,
```
"verify-bounding-box-metadata":
{
  "class-name": "bad",
  "confidence": 0.93,
  "type": "groundtruth/label-verification",
  "job-name": "verify-bounding-boxes",
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256",
  "worker-feedback": [
    {"comment": "The bounding box on the bird is too wide on the right side."},
    {"comment": "The bird on the upper right is not labeled."}
  ]
}

In adjustment tasks, the worker output resembles the worker output of the original task, except it will contain the adjusted values and an adjustment-status property with the value of adjusted or unadjusted to indicate whether an adjustment was made.

See the Output Data (p. 75) page for more examples of the output of different tasks.

Cautions and considerations

Color information requirements for Semantic Segmentation jobs

To properly reproduce color information in verification or adjustment tasks, the tool requires hexadecimal RGB color information in the manifest (e.g. #FFFFFF for white). During the set-up of a Semantic Segmentation verification or adjustment job, the tool will examine the manifest to determine if this information is present. If it cannot find it, an error message is shown and the job set-up cannot be completed.

In prior iterations of the Semantic Segmentation tool, category color information was not output in hexadecimal RGB format to the output manifest. That feature was introduced to the output manifest at the same time the verification and adjustment workflows were introduced. Therefore older output manifests are not compatible with this new workflow.

Filtering your data before starting the job

Amazon SageMaker Ground Truth will process all objects in your input manifest. If you have a partially labeled data set you may want to create a custom manifest using an Amazon S3 Select query on your input manifest. Unlabeled objects will individually fail, but not cause the job to fail, and may incur processing costs. Filtering out objects you don't want verified will also reduce your costs.

There are some filtering tools provided in the console when creating a verification job. If you are creating jobs using the API, make filtering your data part of your workflow where needed.

Security considerations for images

Due to browser security models, some image markup tasks like keypoints, polygons, bounding boxes, and semantic segmentation will require a CORS specification to be added to the Amazon S3 bucket where you store the images. This is necessary to apply prior markup to the images.

Applying CORS to your bucket

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Select the bucket in which you are storing your images.
3. Select the Permissions tab, then CORS configuration.
4. Add the following block of XML and save.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
Use Input and Output Data

The input data that you provide to Amazon SageMaker Ground Truth is sent to your workers for labeling. You choose the data to send to your workers by creating a manifest file that defines the data that requires labeling.

The output data is the result of your labeling job. The output data file contains one entry for each object in the input dataset that specifies the label.

Topics
- Input Data (p. 71)
- Output Data (p. 75)

Input Data

The input data are the data objects that you send to your workforce to be labeled. Each object in the input data is described in a manifest file. Each line in the manifest is an entry containing an object to label. An entry can also contain labels from previous jobs.

Input data and the manifest file must be stored in Amazon Simple Storage Service (Amazon S3). Each has specific storage and access requirements, as follows:

- The S3 bucket that contains the input data must be in the same AWS Region in which you are running Amazon SageMaker Ground Truth. You must give Amazon SageMaker access to the data stored in the S3 bucket so that it can read it. For more information about S3 buckets, see Working with Amazon S3 buckets.

- The manifest file must be in the same AWS Region as the data files, but it doesn't need to be in the same location as the data files. It can be stored in any S3 bucket that is accessible to the AWS Identity and Access Management (IAM) role that you assigned to Ground Truth when you created the labeling job.

The manifest is a UTF-8 encoded file where each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \n\n. Because each line must be a valid JSON object, you can't have unescaped line break characters. For more information about data format, see JSON Lines.

Limits: Each JSON object in the manifest file can be no larger than 100 K characters. No single attribute within an object can be larger than 20,000 characters. Attribute names can't begin with $ (dollar sign).

Each JSON object in the manifest file must contain one of the following keys: source-ref or source. The value of the keys are interpreted as follows:

- source-ref—The source of the object is the Amazon S3 object specified in the value. Use this value when the object is a binary object, such as an image, or when you have text in individual files. You also use the source-ref key for image files for a bounding box or semantic segmentation labeling job.
- source—The source of the object is the value. Use this value when the object is a text value.

The following is an example of a manifest file for files stored in an S3 bucket:

---

71
The following is an example of a manifest file with the input data stored in the manifest:

```json
{"source": "Lorem ipsum dolor sit amet"
{"source": "consectetur adipiscing elit"
...
{"source": "mollit anim id est laborum"
```

You can include other key-value pairs in the manifest file. These pairs are passed to the output file unchanged. This is useful when you want to pass information between your applications. For more information, see Output Data (p. 75).

### Create a Manifest File

You can create a manifest file for your labeling jobs in the Ground Truth console using images, text (txt) files, and comma-separated value (csv) files. Before using the following procedure, ensure that your input images or files are correctly formatted:

- **Image files** – Image files must comply with the size and resolution limits listed in the tables found in Input File Size Quota (p. 73).
- **Text files** – Text data can be stored in one or more .txt files. Each item that you want labeled must be separated by a standard line break.
- **CSV files** – Text data can be stored in one or more .csv files. Each item that you want labeled must be in a separate row.

**To create a manifest file**

1. Store the images or text files that you want to have labeled in an S3 bucket that is in the same Region as your labeling job. You must give access to Amazon SageMaker for the data to be read. For more information about Amazon S3 buckets, see Working with Amazon S3 buckets.
3. Choose Labeling Job.
4. In the Job overview section, under Input dataset location, choose Create manifest file.
5. In Input dataset location, enter the path to the S3 bucket where you data is stored (for example, s3://bucket/path-to-your-objects).
6. Choose the Data type, then choose Create. You see a loading screen while Ground Truth examines the S3 bucket and creates a .manifest file in the S3 location specified above. If you would like to visually inspect the auto-generated manifest file, convert this file into a human-readable format by downloading a copy of the file and changing the file suffix from .manifest to .txt.
7. For images or text object, a green box indicates the number of objects detected. If you're satisfied with the results, choose Use this manifest. If not, modify the image or text files in your S3 bucket and use this procedure to generate a new manifest file.

### Input Data Quotas

Input datasets used in semantic segmentation labeling jobs have a quota of 20,000 items. For all other labeling job types, the dataset size quota is 100,000 items. To request an increase to the quota for labeling jobs other than semantic segmentation jobs, review the procedures in AWS Service Quotas to request a quota increase.
Input image data for active and non-active learning labeling jobs must not exceed size and resolution quotas. *Active learning* refers to labeling job that use automated data labeling. *Non-active learning* refers to labeling jobs that don't use automated data labeling.

**Input File Size Quota**

Input files can't exceed the following size-quotas for both active and non-active learning labeling jobs.

<table>
<thead>
<tr>
<th>Labeling Job</th>
<th>Input File Size Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>12 MB</td>
</tr>
<tr>
<td>Object detection</td>
<td>12 MB</td>
</tr>
<tr>
<td>Semantic segmentation</td>
<td>6 MB</td>
</tr>
<tr>
<td>Image classification label adjustment</td>
<td>12 MB</td>
</tr>
<tr>
<td>Object detection label adjustment</td>
<td>12 MB</td>
</tr>
<tr>
<td>Semantic segmentation label adjustment</td>
<td>6 MB</td>
</tr>
<tr>
<td>Image classification label verification</td>
<td>12 MB</td>
</tr>
<tr>
<td>Object detection label verification</td>
<td>12 MB</td>
</tr>
<tr>
<td>Semantic segmentation label verification</td>
<td>6 MB</td>
</tr>
</tbody>
</table>

**Input Image Resolution Quotas**

Image file resolution refers to the number of pixels in an image, and determines the amount of detail an image holds. Image resolution quotas differ depending on the labeling job type and the Amazon SageMaker built-in algorithm used. The following table lists the resolution quotas for images used in active and non-active learning labeling jobs.

<table>
<thead>
<tr>
<th>Labeling Job</th>
<th>Resolution Quota - Non Active Learning</th>
<th>Resolution Quota - Active Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>3840 x 2160 pixels (4 KB)</td>
</tr>
<tr>
<td>Object detection</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>3840 x 2160 pixels (4 KB)</td>
</tr>
<tr>
<td>Semantic segmentation</td>
<td>3840 x 2160 pixels (4 KB)</td>
<td>1920 x 1080 pixels (1080 p)</td>
</tr>
<tr>
<td>Image classification label adjustment</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>3840 x 2160 pixels (4 KB)</td>
</tr>
<tr>
<td>Object detection label adjustment</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>3840 x 2160 pixels (4 KB)</td>
</tr>
<tr>
<td>Semantic segmentation label adjustment</td>
<td>3840 x 2160 pixels (4 KB)</td>
<td>1920 x 1080 pixels (1080 p)</td>
</tr>
<tr>
<td>Image classification label verification</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>Not available</td>
</tr>
<tr>
<td>Object detection label verification</td>
<td>7680 x 4320 pixels (8 KB)</td>
<td>Not available</td>
</tr>
</tbody>
</table>
Filter and Select Data for Labeling

You can use the Amazon SageMaker console to select a portion of your dataset for labeling. The data must be stored in an Amazon S3 bucket. You have three options:

- Use the full dataset.
- Choose a randomly selected sample of the dataset.
- Specify a subset of the dataset using a query.

The following options are available in the Labeling jobs section of the Amazon SageMaker console after selecting Create labeling job. See Getting started (p. 50) to learn how to Create a labeling job in the console. To configure the dataset that you use for labeling, in the Job overview section, select Additional configuration.

Use the Full Dataset

When you choose to use Full dataset, you must provide a manifest file for your data objects. You can provide the path of the S3 bucket that contains the manifest file or you can use the Amazon SageMaker console to create the file. See Create a Manifest File (p. 72) to learn how to create a manifest file using the console.

Choose a Random Sample

When you want to label a random subset of your data, select Random sample. The dataset is stored in the S3 bucket specified in the Input dataset location field.

After you have specified the percentage of data objects that you want to include in the sample, choose Create subset. Amazon SageMaker randomly picks the data objects for your labeling job. After the objects are selected, choose Use this subset.

Amazon SageMaker creates a manifest file for the selected data objects. It also modifies the value in the Input dataset location field to point to the new manifest file.

Specify a Subset

You can specify a subset of your data objects using an Amazon Simple Storage Service (Amazon S3) SELECT query on the object file names.

The SELECT statement of the SQL query is defined for you. You provide the WHERE clause to specify which data objects should be returned.

For more information about the Amazon S3 SELECT statement, see Selecting Content from Objects.

Choose Create subset to start the selection, and then choose Use this subset to use the selected data.

Amazon SageMaker creates a manifest file for the selected data objects. It also updates the value in the Input dataset location field to point to the new manifest file.
Output Data

The output from a labeling job is placed in the location that you specified in the console or in the call to the CreateLabelingJob operation.

Each line in the output data file is identical to the manifest file with the addition of an attribute and value for the label assigned to the input object. The attribute name for the value is defined in the console or in the call to the CreateLabelingJob operation. You can't use -metadata in the label attribute name. If you are running a semantic segmentation job, the label attribute must end with -ref. For any other type of job, the attribute name can't end with -ref.

The output of the labeling job is the value of the key/value pair with the label. The label and the value overwrites any existing JSON data in the input file with the new value.

For example, the following is the output from an image classification labeling job where the input data files were stored in an Amazon S3 bucket and the label attribute name was defined as "sport". In this example the JSON object is formatted for readability, in the actual output file the JSON object is on a single line. For more information about the data format, see JSON Lines.

```
{
   "source-ref": "S3 bucket location",
   "sport":0,
   "sport-metadata": {
      "class-name": "football",
      "confidence": 0.8,
      "type":"groundtruth/image-classification",
      "job-name": "identify-sport",
      "human-annotated": "yes",
      "creation-date": "2018-10-18T22:18:13.527256"
   }
}
```

The value of the label can be any valid JSON. In this case the label's value is the index of the class in the classification list. Other job types, such as bounding box, have more complex values.

Any key-value pair in the input manifest file other than the label attribute is unchanged in the output file. You can use this to pass data to your application.

The output from a labeling job can be used as the input to another labeling job. You can use this when you are chaining together labeling jobs. For example, you can send one labeling job to determine the sport that is being played. Then you send another using the same data to determine if the sport is being played indoors or outdoors. By using the output data from the first job as the manifest for the second job, you can consolidate the results of the two jobs into one output file for easier processing by your applications.

The output data file is written to the output location periodically while the job is in progress. These intermediate files contain one line for each line in the manifest file. If an object is labeled, the label is included, if the object has not been labeled it is written to the intermediate output file identically to the manifest file.

Output Directories

Ground Truth creates several directories in your Amazon S3 output path. These directories contain the results of your labeling job and other artifacts of the job. The top-level directory for a labeling job is given the same name as your labeling job, the output directories are placed beneath it. For example, if you named your labeling job find-people you output would be in the following directories:

```
s3://bucket/find-people/activelearning
s3://bucket/find-people/annotations
```
Each directories contain the following output:

### Active Learning Directory

The `activelearning` directory is only present when you are using automated data labeling. It contains the input and output validation set for automated data labeling, and the input and output folder for automatically labeled data.

### Annotations Directory

The `annotations` directory contains all of the annotations made by the workforce. These are the responses from individual workers that have not been consolidated into a single label for the data object.

There are three subdirectories in the `annotations` directory.

The first, `worker-response` contains the responses from individual workers. This contains a subdirectory for each iteration, which in turn contains a subdirectory for each data object in that iteration. The annotation for each data object is stored in a timestamped .json file. There may be more than one annotation for each data object in this directory, depending on how many workers you want to annotate each object.

The second, `consolidated-annotation` contains information required to consolidate the annotations in the current batch into labels for your data objects.

The third, `intermediate`, contains the output manifest for the current batch with any completed labels. This file is updated as the label for each data object is completed.

### Inference Directory

The `inference` directory is only present when you are using automated data labeling. This directory contains the input and output files for the Amazon SageMaker batch transform used while labeling data objects.

### Manifest Directory

The `manifest` directory contains the output manifest from your labeling job. There is one subdirectory in the `manifest` directory, `output`. The `output` directory contains the output manifest file for your labeling job. The file is named `output.manifest`.

### Training Directory

The `training` directory is only present when you are using automated data labeling. This directory contains the input and output files used to train the automated data labeling model.

### Confidence Score

Ground Truth calculates a confidence score for each label. A confidence score is a number between 0 and 1 that indicates how confident Ground Truth is in the label. You can use the confidence score to compare labeled data objects to each other, and to identify the least or most confident labels.

You should not interpret the value of the confidence scores as an absolute value, or compare them across labeling jobs. For example, if all of the confidence scores are between 0.98 and 0.998, you should only compare the data objects with each other and not rely on the high confidence scores.

You should not compare the confidence scores of human-labeled data objects and auto-labeled data objects. The confidence scores for humans are calculated using the annotation consolidation function for
the task, the confidence scores for automated labeling are calculated using a model that incorporates object features. The two models generally have different scales and average confidence.

For a bounding box labeling job, Ground Truth calculates a confidence score per box. You can compare confidence scores within one image or across images for the same labeling type (human or auto). You can't compare confidence scores across labeling jobs.

Output Metadata

The output from each job contains metadata about the label assigned to data objects. These elements are the same for all jobs with minor variations. The following example shows the metadata elements.

```
"confidence": 0.93,
"type": "groundtruth/image-classification",
"job-name": "identify-animal-species",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256"
```

The elements have the following meaning:

- **confidence** – The confidence that Ground Truth has that the label is correct. For more information, see Confidence Score (p. 76).
- **type** – The type of classification job. For job types, see Task Type Templates (p. 54).
- **job-name** – The name assigned to the job when it was created.
- **human-annotated** – Indicates whether the data object was labeled by a human or by automated data labeling. For more information, see Using Automated Data Labeling (p. 65).
- **creation-date** – The date and time that the label was created.

Classification Job Output

The following are sample outputs from an image classification job and a text classification job. They includes the label that Ground Truth assigned to the data object, the value for the label, and metadata that describes the labeling task.

In addition to the standard metadata elements, the metadata for a classification job includes the text value of the label's class. For more information, see Image Classification Algorithm (p. 271).

```
{
   "source-ref": "S3 bucket location",
   "species": "0",
   "species-metadata": {
      "class-name": "dog",
      "confidence": 0.93,
      "type": "groundtruth/image-classification",
      "job-name": "identify-animal-species",
      "human-annotated": "yes",
      "creation-date": "2018-10-18T22:18:13.527256"
   }
}
```

```
{
   "source": "a bad day",
   "mood": "1",
   "mood-metadata": {
      "class-name": "sad",
      "confidence": 0.8,
   }
}
```
Bounding Box Job Output

The following is sample output from a bounding box job. For this task, three bounding boxes are returned. The label value contains information about the size of the image, and the location of the bounding boxes.

The \texttt{class\_id} element is the index of the box's class in the list of available classes for the task. The \texttt{class-map} metadata element contains the text of the class.

The metadata has a separate confidence score for each bounding box. The metadata also includes the \texttt{class-map} element that maps the \texttt{class\_id} to the text value of the class. For more information, see Object Detection Algorithm (p. 365).

```json
{
  "source-ref": "S3 bucket location",
  "bounding-box": {
    "image_size": [{ "width": 500, "height": 400, "depth":3}]
  },
  "bounding-box-metadata": {
    "objects": [
      {"confidence": 0.8},
      {"confidence": 0.9},
      {"confidence": 0.9}
    ],
    "class-map": {
      "0": "dog",
      "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:13:527256",
    "job-name": "identify-dogs-and-toys"
  }
}
```

The output of a bounding box adjustment job looks like the following JSON. Note that the original JSON is kept intact and two new jobs are listed, each with "adjust-" prepended to the original attribute's name.

```json
{
  "source-ref": "S3 bucket location",
  "bounding-box": {
    "image_size": [{ "width": 500, "height": 400, "depth":3}]
  }
}
```
In this output, the job's type doesn't change, but an adjustment-status field is added. This field has the value of adjusted or unadjusted. If multiple workers have reviewed the object and at least one adjusted the label, the status is adjusted.
Label Verification Job Output

The output of a bounding box verification job looks much different than the output of a bounding box annotation job. That's because the workers having a different type of task. They're not labeling objects, but evaluating the accuracy of prior labeling, making a judgment, and then providing that judgment and perhaps some comments.

If you are having human workers verify or adjust prior bounding box labels, the output of a verification job would look like the following JSON.

```json
{
  "source-ref": "S3 bucket location",
  "bounding-box": {
    "image_size": [{  "width": 500,  "height": 400,  "depth": 3}],
    "annotations": [
      {"class_id": 0,  "left": 111,  "top": 134,  "width": 61,  "height": 128},
      {"class_id": 5,  "left": 161,  "top": 250,  "width": 30,  "height": 30},
      {"class_id": 5,  "left": 20,  "top": 20,  "width": 30,  "height": 30}
    ],
  },
  "bounding-box-metadata": {
    "objects": [{"confidence": 0.8}, {"confidence": 0.9}, {"confidence": 0.9}],
    "class-map": {
      "0": "dog",
      "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "identify-dogs-and-toys"
  },
  "verify-bounding-box": "1",
  "verify-bounding-box-metadata": {
    "class-name": "bad",
    "confidence": 0.93,
    "type": "groundtruth/label-verification",
    "job-name": "verify-bounding-boxes",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "worker-feedback": [{
      "comment": "The bounding box on the bird is too wide on the right side.",
      "comment": "The bird on the upper right is not labeled."
    ]
  }
}
```

Although the type on the original bounding box output was groundtruth/object-detection, the new type is groundtruth/label-verification. Also note that the worker-feedback array provides worker comments. If the worker doesn't provide comments, the empty fields are excluded during consolidation.
Semantic Segmentation Job Output

The following is the output from a semantic segmentation labeling job. The value of the label for this job is a reference to a PNG file in an S3 bucket.

In addition to the standard elements, the metadata for the label includes a color map that defines which color was used to label the image, the class name associated with the color, and the confidence score for each color. For more information, see Semantic Segmentation Algorithm (p. 402).

```json
{
  "source-ref": "S3 bucket location",
  "city-streets-ref": "S3 bucket location",
  "city-streets-metadata": {
    "internal-color-map": {
      "0": {
        "class-name": "BACKGROUND",
        "confidence": 0.9,
        "hex-color": "#ffffff"
      },
      "1": {
        "class-name": "buildings",
        "confidence": 0.9,
        "hex-color": "#2acf59"
      },
      "2": {
        "class-name": "road",
        "confidence": 0.9,
        "hex-color": "#f28333"
      }
    },
    "type": "groundtruth/semantic-segmentation",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "label-city-streets",
  },
  "verify-city-streets": "1",
  "verify-city-streets-metadata": {
    "class-name": "bad",
    "confidence": 0.93,
    "type": "groundtruth/label-verification",
    "job-name": "verify-city-streets",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "worker-feedback": [
      {"comment": "The mask on the leftmost building is assigned the wrong side of the road.","},
      {"comment": "The curb of the road is not labeled but the instructions say otherwise."}
    ]
  }
}
```

Confidence is scored on a per-image basis. Confidence scores are the same across all classes within an image.

The output of a semantic segmentation adjustment job looks similar to the following JSON.

```json
{
  "source-ref": "S3 bucket location",
  "city-streets-ref": "S3 bucket location",
  "city-streets-metadata": {
    "internal-color-map": {
```
After you create an augmented manifest file, you can use it in a training job. See object_detection_augmented_manifest_training.ipynb for a demonstration of using of an "augmented manifest" to train an object detection machine learning model with AWS SageMaker. For more information, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

Creating Instruction Pages

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console or you can create your own. The instructions are shown to the worker on the page where they complete their labeling task.

There are two kinds of instructions:

- **Short instructions**—instructions that are shown on the same webpage where the worker completes their task. These instructions should provide an easy reference to show the worker the correct way to label an object.
• *Full instructions*—instructions that are shown on a dialog box that overlays the page where the worker completes their task. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

Create instructions in the console when you are creating your labeling job. Start with the existing instructions for the task and use the editor to modify them to suit your labeling job.

**Short Instructions**

Short instructions appear on the same webpage that workers use to label your data object. For example, the following is the editing page for a bounding box task. The short instructions panel is on the left.

Keep in mind that a worker will only spend seconds looking at the short instructions. Workers must be able to scan and understand your information quickly. In all cases it should take less time to understand the instructions than it takes to complete the task. Keep these points in mind:

• Your instructions should be clear and simple.
• Pictures are better than words. Create a simple illustration of your task that your workers can immediately understand.
• If you must use words, use short, concise examples.
• Your short instructions are more important than your full instructions.
The Amazon SageMaker Ground Truth console provides an editor so that you can create your short instructions. Replace the placeholder text and images with instructions for your task. Preview the worker's task page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

### Full Instructions

You can provide additional instructions for your workers in a dialog box that overlays the page where workers label your data objects. Use full instructions to explain more complex tasks and to show workers the proper way to label edge cases or other difficult objects.

You can create full instructions using an editor in the Ground Truth console. As with quick instructions, keep the following in mind:

- Workers will want detailed instruction the first few times that the complete your task. Any information that they *must* have should be in the quick instructions.
- Pictures are more important than words.
- Text should be concise.
- Full instructions should supplement the short instructions. Don't repeat information that appears in the short instructions.

The Ground Truth console provides an editor so that you can create your full instructions. Replace the placeholder text and images with instructions for your task. Preview the full instruction page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

### Add example images to your instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions:

- Place the cursor where the image should go in the instructions editor.
- Click the image icon in the editor toolbar.
- Enter the URL of your image.

If your instruction image in Amazon S3 is not publicly accessible:

- As the image URL, enter: `{{ 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access }}`.
- This renders the image URL with a short-lived, one-time access code appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview.

### Creating Custom Labeling Workflows

This document guides you through the process of setting up a workflow with a custom labeling template. For more information about starting a labeling job, see Getting started (p. 50). In that section, when you choose the Task type, select **Custom labeling task**, and then follow this section's instructions to configure it. For a repository of demo templates for a variety of labeling job task types, see Amazon SageMaker Ground Truth Sample Task UIs.

**Next**

*Step 1: Setting up your workforce (p. 85)*
Step 1: Setting up your workforce

In this step you use the console to establish which worker type to use and make the necessary sub-
selections for the worker type. It assumes you have already completed the steps up to this point in the
Getting started (p. 50) section and have chosen the Custom labeling task as the Task type.

To configure your workforce.

1. First choose an option from the Worker types. There are three types currently available:
   - **Public** uses an on-demand workforce of independent contractors, powered by Amazon Mechanical
     Turk. They are paid on a per-task basis.
   - **Private** uses your employees or contractors for handling data that needs to stay within your
     organization.
   - **Vendor** uses third party vendors that specialize in providing data labeling services, available via
     the AWS Marketplace.

2. If you choose the Public option, you are asked to set the number of workers per dataset object.
   Having more than one worker perform the same task on the same object can help increase the
   accuracy of your results. The default is three. You can raise or lower that depending on the accuracy
   you need.

   You are also asked to set a price per task by using a drop-down menu. The menu recommends price
   points based on how long it will take to complete the task.

   The recommended method to determine this is to first run a short test of your task with a private
   workforce. The test provides a realistic estimate of how long the task takes to complete. You can
   then select the range your estimate falls within on the Price per task menu. If your average time is
   more than 5 minutes, consider breaking your task into smaller units.

Next

Step 2: Creating your custom labeling task template (p. 85)

Step 2: Creating your custom labeling task template

Topics

- Starting with a base template (p. 85)
- Developing templates locally (p. 86)
- Using External Assets (p. 86)
- Track your variables (p. 86)
- A simple sample (p. 86)
- Adding automation with Liquid (p. 88)
- End-to-end demos (p. 90)
- Next (p. 91)

Starting with a base template

To get you started, the Task type starts with a drop-down menu listing a number of our more common
task types, plus a custom type. Choose one and the code editor area will be filled with a sample template
for that task type. If you prefer not to start with a sample, choose Custom HTML for a minimal template
skeleton.
If you've already created a template, upload the file directly using the **Upload file** button in the upper right of the task setup area or paste your template code into the editor area.

**Developing templates locally**

While you need to be in the console to test how your template will process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding this code to the top of your HTML file.

**Example**

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you want to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this will not parse your variables. You may want to replace them with sample content while developing locally.

**Using External Assets**

Amazon SageMaker Ground Truth custom templates allow external scripts and style sheets to be embedded.

**Example**

```html
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding types for remote scripts: `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets: `text/css;CHARSET=UTF-8`.

**Track your variables**

In the process of building the sample below, there will be a step that adds variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you will need to make sure you're aware of the variables it already uses. When you create your pre- annotation AWS Lambda script, its output will need to contain values for any of those variables you choose to keep.

The values you use for the variables can come from your manifest file. All the key-value pairs in your data object are provided to your pre- annotation Lambda. If it's a simple pass-through script, matching keys for values in your data object to variable names in your template is the easiest way to pass those values through to the tasks forms your workers see.

**A simple sample**

All tasks begin and end with the `<crowd-form> </crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them.
For a simple tweet-analysis task, use the `<crowd-classifier>` element. It requires the following attributes:

- **name** - the variable name to use for the result in the form output.
- **categories** - a JSON formatted array of the possible answers.
- **header** - a title for the annotation tool

As children of the `<crowd-classifier>` element, you must have three regions.

- **<classification-target>** - the text the worker will classify based on the options specified in the `categories` attribute above.
- **<full-instructions>** - instructions that are available from the "View full instructions" link in the tool. This can be left blank, but it is recommended that you give good instructions to get better results.
- **<short-instructions>** - a more brief description of the task that appears in the tool's sidebar. This can be left blank, but it is recommended that you give good instructions to get better results.

A simple version of this tool would look like this.

**Example of using crowd-classifier**

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="["positive","negative","neutral","unclear"]"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      My favorite football team won today!
      Bring on the division finals!
    </classification-target>

    <full-instructions header="Sentiment Analysis Instructions">
      Try to determine the sentiment the author of the tweet is trying to express. If none seem to match, choose "cannot determine."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

You can copy and paste the code into the editor in the Ground Truth labeling job creation workflow to preview the tool, or try out a demo of this code on CodePen.
Adding automation with Liquid

Our custom template system uses Liquid for automation. It is an open source inline markup language. For more information and documentation, visit the Liquid homepage.

The most common use of Liquid will be to parse the data coming from your pre-annotation Lambda and pull out the relevant variables to create the task. In Liquid, the text between single curly braces and percent symbols is an instruction or "tag" that creates control flow. Text between double curly braces is a variable or "object" which outputs its value.

The taskInput object returned by your Pre-annotation Lambda (p. 102) will be available as the task.input object in your templates.

The properties in your manifest's data objects are passed into your Pre-annotation Lambda (p. 102) as the event.dataObject. A simple pass-through script simply returns that object as the taskInput object. You would represent values from your manifest as variables as follows.

Example Manifest data object

```json
{
  "source": "This is a sample text for classification",
  "labels": [ "angry", "sad", "happy", "inconclusive" ],
  "header": "What emotion is the speaker feeling?"
}
```

Example Sample HTML using variables

```html
<crowd-classifier
    name='tweetFeeling'
    categories='{{ task.input.labels | to_json }}'
    header='{{ task.input.header }}' />
```
Note the addition of " | to_json" to the labels property above. That's a filter to turn the array into a JSON representation of the array. Variable filters are explained next.

Variable filters

In addition to the standard Liquid filters and actions, Ground Truth offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, then specifying a filter name. Filters can be chained in the form of:

Example

```{{ <content> | <filter> | <filter> }}```

Autoescape and explicit escape

By default, inputs will be HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the escape filter to make it more obvious to someone reading the source of your template that the escaping is being done.

**escape_once**

escape_once ensures that if you've already escaped your code, it doesn't get re-escaped on top of that. For example, so that & doesn't become &amp;&amp;.

**skip_autoescape**

skip_autoescape is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

*Use skip_autoescape sparingly*

The best practice in templates is to avoid passing in functional code or markup with skip_autoescape unless you are absolutely sure you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a Cross Site Scripting attack.

**to_json**

to_json will encode what you feed it to JSON (JavaScript Object Notation). If you feed it an object, it will serialize it.

**grant_read_access**

grant_read_access takes an S3 URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible.

Example of the filters

Input

```auto-escape: {{ "Have you read 'James & the Giant Peach'?' }}
explicit escape: {{ "Have you read 'James & the Giant Peach'?' | escape }}```
Example
Output

auto-escape: Have you read 'James & the Giant Peach'?
explicit escape: Have you read 'James & the Giant Peach'?
explicit escape_once: Have you read 'James & the Giant Peach'?
skip_autoescape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?<access token and other params>

Example of an automated classification template.

To automate the simple text classification sample, replace the tweet text with a variable.

The text classification template is below with automation added. The changes/additions are highlighted in bold.

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>

<crowd-classifier
name="tweetFeeling"
categories=
['positive', 'negative', 'neutral', 'cannot determine']
header="Which term best describes this tweet?"
>
  <classification-target>
    {{ task.input.source }}
  </classification-target>

  <full-instructions header="Analyzing a sentiment">
    Try to determine the feeling the author of the tweet is trying to express.
    If none seem to match, choose "other."
  </full-instructions>

  <short-instructions>
    Pick the term best describing the sentiment of the tweet.
  </short-instructions>

</crowd-classifier>
</crowd-form>

The tweet text that was in the prior sample is now replaced with an object. The entry.taskInput object uses source (or another name you specify in your pre-annotation Lambda) as the property name for the text and it is inserted directly in the HTML by virtue of being between double curly braces.

End-to-end demos

You can view the following end-to-end demos which include sample Lambdas:

- Demo Template: Annotation of Images with crowd-bounding-box (p. 91)
- Demo Template: Labeling Intents with crowd-classifier (p. 95)
Next

Step 3: Processing with AWS Lambda (p. 102)

Demo Template: Annotation of Images with crowd-bounding-box

When you chose to use a custom template as your task type in the Amazon SageMaker Ground Truth console, you reach the Custom labeling task panel. There you can choose from multiple base templates. The templates represent some of the most common tasks and provide a sample to work from as you create your customized labeling task's template. If you are not using the console, or as an additional recourse, see Amazon SageMaker Ground Truth Sample Task UIs for a repository of demo templates for a variety of labeling job task types.

This demonstration works with the BoundingBox template. The demonstration also works with the AWS Lambda functions needed for processing your data before and after the task. In the Github repository above, to find templates that work with AWS Lambda functions, look for {{ task.input.<property name> }} in the template.

Topics

- Starter Bounding Box custom template (p. 91)
- Your own Bounding Box custom template (p. 92)
- Your manifest file (p. 93)
- Your pre-annotation Lambda function (p. 93)
- Your post-annotation Lambda function (p. 94)
- The output of your labeling job (p. 95)

Starter Bounding Box custom template

This is the starter bounding box template that is provided.

```html
<full-instructions header="Bounding Box Instructions" >
<p>Use the bounding box tool to draw boxes around the requested target of interest:</p>
<ol>
  <li>Draw a rectangle using your mouse over each instance of the target.</li>
  <li>Make sure the box does not cut into the target, leave a 2 – 3 pixel margin</li>
  <li>When targets are overlapping, draw a box around each object, include all contiguous parts of the target in the box. Do not include parts that are completely overlapped by another object.</li>
  <li>Do not include parts of the target that cannot be seen, even though you think you can interpolate the whole shape of the target.</li>
</ol>
```

91
Avoid shadows, they’re not considered as a part of the target.

If the target goes off the screen, label up to the edge of the image.

<!-- The <short-instructions> tag allows you to specify instructions that are displayed in the left hand side of the task interface. It is a best practice to provide good and bad examples in this section for quick reference. -->

Use the bounding box tool to draw boxes around the requested target of interest.

The custom templates use the Liquid template language, and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named taskInput and that object's properties can be accessed as {{ task.input.<property name> }} in your template.

Your own Bounding Box custom template

As an example, assume you have a large collection of animal photos in which you know the kind of animal in an image from a prior image-classification job. Now you want to have a bounding box drawn around it.

In the starter sample, there are three variables: taskObject, header, and labels.

Each of these would be represented in different parts of the bounding box.

- **taskObject** is an HTTP(S) URL or S3 URI for the photo to be annotated. The added | grant_read_access is a filter that will convert an S3 URI to an HTTPS URL with short-lived access to that resource. If you're using an HTTP(S) URL, it's not needed.
- **header** is the text above the photo to be labeled, something like "Draw a box around the bird in the photo."
- **labels** is an array, represented as ['item1', 'item2', ...]. These are labels that can be assigned by the worker to the different boxes they draw. You can have one or many.

Each of the variable names come from the JSON object in the response from your pre-annotation Lambda. The names above are merely suggested. Use whatever variable names make sense to you and will promote code readability among your team.

Only use variables when necessary

If a field will not change, you can remove that variable from the template and replace it with that text, otherwise you have to repeat that text as a value in each object in your manifest or code it into your pre-annotation Lambda function.

Example: Final Customized Bounding Box Template

To keep things simple, this template will have one variable, one label, and very basic instructions. Assuming your manifest has an "animal" property in each data object, that value can be re-used in two parts of the template.
Note the re-use of \{{ task.input.animal \}} throughout the template. If your manifest had all of the animal names beginning with a capital letter, you could use \{{ task.input.animal | downcase \}}, incorporating one of Liquid's built-in filters in sentences where it needed to be presented lowercase.

Your manifest file

Your manifest file should provide the variable values you're using in your template. You can do some transformation of your manifest data in your pre-annotation Lambda, but if you don't need to, you maintain a lower risk of errors and your Lambda will run faster. Here's a sample manifest file for the template.

```json
{
"source-ref": "<S3 image URI>", "animal": "horse"
"source-ref": "<S3 image URI>", "animal": "bird"
"source-ref": "<S3 image URI>", "animal": "dog"
"source-ref": "<S3 image URI>", "animal": "cat"
}
```

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process your manifest entries and pass them to the template engine.

**Naming your Lambda function**

The best practice in naming your function is to use one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation functions.

When you're using the console, if you have AWS Lambda functions that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic example, you're just passing through the information from the manifest without doing any additional processing on it. This sample pre-annotation function is written for Python 3.7.

```python
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```
The JSON object from your manifest will be provided as a child of the event object. The properties inside the taskInput object will be available as variables to your template, so simply setting the value of taskInput to event['dataObject'] will pass all the values from your manifest object to your template without having to copy them individually. If you wish to send more values to the template, you can add them to the taskInput object.

Your post-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer consolidation and scoring as it comes in, you can apply the scoring and/or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

Provide permissions to your post-annotation Lambda

The annotation data will be in a file designated by the s3Uri string in the payload object. To process the annotations as they come in, even for a simple pass through function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files. In the Console page for creating your Lambda, scroll to the Execution role panel. Select Create a new role from one or more templates. Give the role a name. From the Policy templates drop-down, choose Amazon S3 object read-only permissions. Save the Lambda and the role will be saved and selected.

The following sample is in Python 2.7.

```python
import json
import boto3
from urlparse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket=parsed_url.netloc, Key=parsed_url.path[1:])
    filecont = textFile['Body'].read()
    annotations = json.loads(filecont);

    for dataset in annotations:
        for annotation in dataset['annotations']:
            new_annotation = json.loads(annotation['annotationData']['content'])
            label = {
                'datasetObjectId': dataset['datasetObjectId'],
                'consolidatedAnnotation': {
                    'content': {
                        event['labelAttributeName']: {
                            'workerId': annotation['workerId'],
                            'boxesInfo': new_annotation,
                            'imageSource': dataset['dataObject']
                        }
                    }
                }
            }
            consolidated_labels.append(label)

    return consolidated_labels
```

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through. What you send back will be an object meeting the API contract (p. 102).
The output of your labeling job

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For a bounding box task, the output you find in the output manifest will look a bit like the demo below. The example has been cleaned up for printing. The actual output will be a single line per record.

**Example : JSON in your output manifest**

```json
{
  "source-ref" : "<URL>",
  "label attribute name" : {
    "workerId" : "<URL>",
    "imageSource" : "<image URL>",
    "boxesInfo" : {
      "boundingBox" : {
        "boundingBoxes" : [
          {
            "height" : 878,
            "label" : "bird",
            "left" : 208,
            "top" : 6,
            "width" : 809
          }
        ],
        "inputImageProperties" : {
          "height" : 924,
          "width" : 1280
        }
      }
    },
    "label attribute name" - metadata : {
      "type" : "groundTruth/custom",
      "job_name" : "<Labeling job name>",
      "human-annotated" : "yes"
    },
    "animal" : "bird"
}
```

Note how the additional `animal` attribute from your original manifest is passed to the output manifest on the same level as the `source-ref` and labeling data. Any properties from your input manifest, whether they were used in your template or not, will be passed to the output manifest.

This should help you create your own custom template.

**Demo Template: Labeling Intents with crowd-classifier**

If you choose a custom template, you'll reach the Custom labeling task panel. There you can select from multiple starter templates that represent some of the more common tasks. The templates provide a starting point to work from in building your customized labeling task's template.

In this demonstration, you work with the Intent Detection template, which uses the crowd-classifier (p. 151) element, and the AWS Lambda functions needed for processing your data before and after the task.

**Topics**

- Starter Intent Detection custom template (p. 95)
- Your Intent Detection custom template (p. 96)
- Your pre-annotation Lambda function (p. 99)
- Your post-annotation Lambda function (p. 100)
- Your labeling job output (p. 101)

**Starter Intent Detection custom template**

This is the intent detection template that is provided as a starting point.

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```
The custom templates use the Liquid template language, and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

**Your Intent Detection custom template**

In the starter template, there are two variables: the `task.input.labels` property in the `crowd-classifier` element opening tag and the `task.input.utterance` in the `classification-target` region's content.

Unless you need to offer different sets of labels with different utterances, avoiding a variable and just using text will save processing time and creates less possibility of error. The template used in this demonstration will remove that variable, but variables and filters like `to_json` are explained in more detail in the crowd-bounding-box demonstration article.

**Styling Your Elements**

Two parts of these custom elements that sometimes get overlooked are the `<full-instructions>` and `<short-instructions>` regions. Good instructions generate good results.

In the elements that include these regions, the `<short-instructions>` appear automatically in the "Instructions" pane on the left of the worker's screen. The `<full-instructions>` are linked from the "View full instructions" link near the top of that pane. Clicking the link opens a modal pane with more detailed instructions.

You can not only use HTML, CSS, and JavaScript in these sections, you are encouraged to if you believe you can provide a strong set of instructions and examples that will help workers complete your tasks with better speed and accuracy.
Example Try out a sample with JSFiddle

Try out an example `<crowd-classifier>` task. The example is rendered by JSFiddle, therefore all the template variables are replaced with hard-coded values. Click the "View full instructions" link to see a set of examples with extended CSS styling. You can fork the project to experiment with your own changes to the CSS, adding sample images, or adding extended JavaScript functionality.

Example: Final Customized Intent Detection Template

This uses the example `<crowd-classifier>` task, but with a variable for the `<classification-target>`. If you are trying to keep a consistent CSS design among a series of different labeling jobs, you can include an external stylesheet using a `<link rel...>` element the same way you’d do in any other HTML document.

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<form>
  <crowd-classifier
    name="intent"
    categories="[\'buy', \'eat', \'watch', \'browse', \'leave\']"
    header="Pick the most relevant intent expressed by the text below"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>
    <full-instructions header="Emotion Classification Instructions">
      <p>In the statements and questions provided in this exercise, what category of action is the speaker interested in doing?</p>
      <table>
        <tr>
          <th>Example Utterance</th>
          <th>Good Choice</th>
        </tr>
        <tr>
          <td>example utterance</td>
          <td>example good choice</td>
        </tr>
      </table>
    </full-instructions>
  </crowd-classifier>
</form>
```
<td>When is the Seahawks game on?</td>
<td>
eat<br>
<greenbg>watch</greenbg>
<botchoice>browse</botchoice>
</td>
</tr>
<tr>
<th>Example Utterance</th>
<th>Bad Choice</th>
</tr>
<tr>
<td>When is the Seahawks game on?</td>
<td>
buy<br>
<greenbg>eat</greenbg>
<botchoice>watch</botchoice>
</td>
</tr>
Example: Your manifest file

If you are preparing your manifest file manually for a text-classification task like this, have your data formatted in the following manner.

```json
{"source": "Roses are red"}
{"source": "Violets are Blue"}
{"source": "Ground Truth is the best"}
{"source": "And so are you"}
```

This differs from the manifest file used for the "Demo Template: Annotation of Images with crowd-bounding-box (p. 91)" demonstration in that source-ref was used as the property name instead of source. The use of source-ref designates S3 URIs for images or other files that must be converted to HTTP. Otherwise, source should be used like it is with the text strings above.

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda that can be called to process your manifest entries and pass them to the template engine.

This Lambda function is required to have one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction.

This applies to both your pre-annotation and post-annotation Lambdas.

When you're using the console, if you have Lambdas that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.
In this very basic sample, where you have only one variable, it's primarily a pass-through function. Here's a sample pre-labeling Lambda using Python 3.7.

```python
import json
def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

The `dataObject` property of the event contains the properties from a data object in your manifest.

In this demonstration, which is a simple pass through, you just pass that straight through as the `taskInput` value. If you add properties with those values to the `event['dataObject']` object, they will be available to your HTML template as Liquid variables with the format `{{ task.input.<property name> }}`.

**Your post-annotation Lambda function**

As part of the job set up, provide the ARN of an Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer-consolidation and scoring as data comes in, you can apply the scoring or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

**Set permissions for your post-annotation Lambda function**
The annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files.
In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is for Python 3.7.

```python
import json
import boto3
from urllib.parse import urlparse
def lambda_handler(event, context):
    consolidated_labels = []
    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
filecont = textFile['Body'].read()
annotations = json.loads(filecont);
for dataset in annotations:
    for annotation in dataset['annotations']:
        new_annotation = json.loads(annotation['annotationData']['content'])
        label = {
            'datasetObjectId': dataset['datasetObjectId'],
            'consolidatedAnnotation': {
                'content': {
                    event['labelAttributeName']: {
                        'workerId': annotation['workerId'],
                        'result': new_annotation,
                        'labeledContent': dataset['dataObject']
                    }
                }
            }
        }
        consolidated_labels.append(label)
```

100
Your labeling job output

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through.

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named manifests.

For an intent detection task, the output in the output manifest will look a bit like the demo below. The example has been cleaned up and spaced out to be easier for humans to read. The actual output will be more compressed for machine reading.

Example : JSON in your output manifest

```json
[
  {
    "datasetObjectId":"<Number representing item's place in the manifest>",
    "consolidatedAnnotation":
    {
      "content":
      {
        "<name of labeling job>":
        {
          "workerId":"private.us-east-1.XXXXXXXXXXXXXXXXXXXXX",
          "result":
          {
            "intent":
            {
              "label":"<label chosen by worker>"
            }
          },
          "labeledContent":
          {
            "content": "<text content that was labeled>"
          }
        }
      }
    },
    "datasetObjectId":"<Number representing item's place in the manifest>",
    "consolidatedAnnotation":
    {
      "content":
      {
        "<name of labeling job>":
        {
          "workerId":"private.us-east-1.6UDLPKQZHYWGSCA4MBJBB7FWE",
          "result":
          {
            "intent":
            {
              "label": "<label chosen by worker>"
            }
          },
          "labeledContent":
          {
            "content": "<text content that was labeled>"
          }
        }
      }
    }
  }
]
This should help you create and use your own custom template.

**Step 3: Processing with AWS Lambda**

In this step, you set which AWS Lambda functions to trigger on each dataset object prior to sending it to workers and which function will be used to process the results once the task is submitted. These functions are required.

You will first need to visit the AWS Lambda console or use AWS Lambda's APIs to create your functions. The AmazonSageMakerFullAccess policy is restricted to invoking AWS Lambda functions with one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation Lambdas. If you choose to use names without those strings, you must explicitly provide lambda:InvokeFunction permission to the IAM role used for creating the labeling job.

Select your lambdas from the **Lambda functions** section that comes after the code editor for your custom HTML in the Ground Truth console.

If you need an example, there is an end-to-end demo, including Python code for the Lambdas, in the "Demo Template: Annotation of Images with crowd-bounding-box (p. 91)" document.

**Pre-annotation Lambda**

Before a labeling task is sent to the worker, your AWS Lambda function will be sent a JSON formatted request to provide details.

**Example of a Pre-annotation request**

```json
{
  "version": "2018-10-16",
  "labelingJobArn": <labelingJobArn>
  "dataObject" : {
    "source-ref": "s3://mybucket/myimage.png"
  }
}
```

The `dataObject` will contain the JSON formatted properties from your manifest's data object. For a very basic image annotation job, it might just be a `source-ref` property specifying the image to be annotated. The JSON line objects in your manifest can be up to 100 kilobytes in size and contain a variety of data.

In return, Ground Truth will require a response formatted like this:

**Example of expected return data**

```json
{
  "taskId": <json object>,
  "isHumanAnnotationRequired": <boolean> # Optional
}
```

That `<json object>` may be a bit deceiving. It needs to contain all the data your custom form will need. If you’re doing a bounding box task where the instructions stay the same all the time, it may just
be the HTTP(S) or S3 resource for your image file. If it's a sentiment analysis task and different objects may have different choices, it would be the object reference as a string and the choices as an array of strings.

**Implications of `isHumanAnnotationRequired`**

This value is optional because it will default to `true`. The primary use case for explicitly setting it is when you want to exclude this data object from being labeled by human workers.

If you have a mix of objects in your manifest, with some requiring human annotation and some not needing it, you can include a `isHumanAnnotationRequired` value in each data object. You can then use code in your pre-annotation Lambda to read the value from the data object and set the value in your Lambda output.

**The pre-annotation Lambda runs first**

Before any tasks are available to workers, your entire manifest will be processed into an intermediate form, using your Lambda. This means you won't be able to change your Lambda part of the way through a labeling job and see that have an impact on the remaining tasks.

**Post-annotation Lambda**

When all workers have annotated the data object or when `TaskAvailabilityLifetimeInSeconds` has been reached, whichever comes first, Ground Truth will send those annotations to your Post-annotation Lambda. This Lambda is generally used for Annotation Consolidation (p. 63). The request object will come in like this:

```json
{
  "version": "2018-10-16",
  "labelingJobArn": <labelingJobArn>,
  "labelCategories": [<string>],
  "labelAttributeName": <string>,
  "roleArn": "string",
  "payload": {
    "s3Uri": <string>
  }
}
```

**Note**

If no worker work on the data object and `TaskAvailabilityLifetimeInSeconds` has been reached, data object will be marked as failed and not included as part of post annotation lambda invocation.

**Post-labeling task Lambda permissions**

The actual annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign the necessary permissions to your Lambda to read files from your S3 bucket.

In the Console page for creating your Lambda, scroll to the Execution role panel. Select Create a new role from one or more templates. Give the role a name. From the Policy templates drop-down, choose Amazon S3 object read-only permissions. Save the Lambda and the role will be saved and selected.

**Example of an annotation data file**

```json
[
  {
    "datasetObjectId": <string>,
    "dataObject": {
      "s3Uri": <string>,
      "content": <string>
```
Essentially, all the fields from your form will be in the `content` object. At this point you can start running data consolidation algorithms on the data, using an AWS database service to store results. Or you can pass some processed/optimized results back to Ground Truth for storage in your consolidated annotation manifests in the S3 bucket you specify for output during the configuration of the labeling job.

In return, Ground Truth will require a response formatted like this:

### Example of expected return data

```
[
  {
    "datasetObjectId": "<string>",
    "consolidatedAnnotation": {
      "content": {
        "<labelAttributeName>": {
          # ... label content
        }
      }
    }
  },
  {
    "datasetObjectId": "<string>",
    "consolidatedAnnotation": {
      "content": {
        "<labelAttributeName>": {
          # ... label content
        }
      }
    }
  }
]
```

At this point, all the data you're sending to your S3 bucket, other than the `datasetObjectId` will be in the `content` object.

That will result in an entry in your job's consolidation manifest like this:

### Example of label format in output manifest

```
{
  "source-ref"/"source" : "<s3uri or content>",
  "<labelAttributeName>" : {
    # ... label content from you
  },
  "<labelAttributeName>-metadata": {
    # This will be added by Ground Truth
    "job_name": "<labelingJobName>",
    "type": "groundTruth/custom",
    "human-annotated": "yes",
    "<labelAttributeName>-job-name": ...
  }
}
```
Because of the potentially complex nature of a custom template and the data it collects, Ground Truth does not offer further processing of the data or insights into it.

Next

Custom Workflows via the API (p. 105)

Custom Workflows via the API

When you have created your custom UI template (Step 2) and processing Lambda functions (Step 3), you should place the template in an Amazon S3 bucket with a file name format of: `<FileName>.liquid.html`.

Use the `CreateLabelingJob` (p. 897) action to configure your task. You'll use the location of a custom template (Step 2: Creating your custom labeling task template (p. 85)) stored in a `<filename>.liquid.html` file on S3 as the value for the `UiTemplateS3Uri` field in the `UiConfig` (p. 1568) object within the `HumanTaskConfig` (p. 1372) object.

For the AWS Lambda tasks described in Step 3: Processing with AWS Lambda (p. 102), the post-annotation task's ARN will be used as the value for the `AnnotationConsolidationLambdaArn` field, and the pre-annotation task will be used as the value for the `PreHumanTaskLambdaArn`.

Using Amazon Augmented AI for Human Review

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

When you use AI applications such as Amazon Rekognition, Amazon Textract, or your custom machine learning (ML) models you can use Amazon Augmented AI to get human review of low confidence or a random sample of predictions.

What is Amazon Augmented AI?

Amazon Augmented AI (Amazon A2I) makes it easy to build the workflows required for human review of ML predictions. Amazon A2I brings human review to all developers, removing the undifferentiated heavy lifting associated with building human review systems or managing large numbers of human reviewers.

Many machine learning applications require humans to review low-confidence predictions to ensure the results are correct. For example, extracting information from scanned mortgage application forms can require human review in some cases due to low-quality scans or poor handwriting. But building human review systems can be time-consuming and expensive because it involves implementing complex processes or workflows, writing custom software to manage review tasks and results, and in many cases, managing large groups of reviewers.

Amazon A2I makes it easy to build and manage human reviews for machine learning applications. Amazon A2I provides built-in human review workflows for common machine learning use cases, such as content moderation and text extraction from documents, which allows predictions from Amazon Rekognition and Amazon Textract to be reviewed easily. You can also create your own workflows for ML models built on Amazon SageMaker or any other tools. Using Amazon A2I, you can allow human reviewers to step in when a model is unable to make a high-confidence prediction or to audit its predictions on an ongoing basis.
Get Started with Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Augmented AI (Amazon A2I) helps you integrate human intelligence into AI/ML workflows. With Amazon A2I, you can let AI handle straight-forward data and invoke human reviewers only when their skills are needed. You are able to set conditions, such as confidence thresholds, on inferences when using Amazon A2I with Amazon Textract and Amazon Rekognition. When these conditions are met, the inference request is sent to human review. This entire process is called a human review workflow or flow definition.

Prerequisites

You can add a Amazon A2I human review workflow to Amazon Textract, Amazon Rekognition, and custom data analysis jobs using both the Amazon SageMaker console and an API. To create a human review workflow, you need the following:

- One or more Amazon S3 buckets in the same AWS Region as the workflow for your input and output data. To create a bucket, follow the instructions in Create a Bucket in the Amazon Simple Storage Service Console User Guide.
- An IAM role with required permissions. For more information, see Permissions and Security in Amazon Augmented AI (p. 121).
- You're prompted to select a public, private, or vendor workforce for your human review workflows. If you plan to use a private workforce, you need to set one up ahead of time in the same AWS Region.
Create a Human Review Workflow (Console)

Create a flow definition in the console to incorporate a human data review into your Amazon Rekognition, Amazon Textract, and custom task workflow. This topic assumes that you have completed the flow definition as described in Prerequisites (p. 106).

To create a human review workflow in the console

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In Augmented AI, choose Human review workflows and then choose Create human review workflow.
3. In Overview, do the following:
   a. In Name, enter a unique workflow name.
   b. In S3 location for output, enter the S3 bucket where the human review results should be stored. This bucket must be located in the same AWS Region as the workflow.
   c. Choose an IAM role with your required permissions attached. For more information about required permissions for each Augmented AI task type, see Permissions and Security in Amazon Augmented AI (p. 121).
4. In Task type, choose the task type that you want the human worker to perform.
5. For Amazon Rekognition and Amazon Textract task types, specify the conditions that you want to invoke the human review.
   - For Amazon Rekognition image moderation tasks, choose an inference confidence score threshold interval at which point you want to trigger a human review.
   - For Amazon Textract tasks, you can trigger a human review when specific form keys are missing or when form key detection confidence is low. You can also trigger a human review if, after

Important
Use the Amazon Mechanical Turk workforce only to process data that is public or has been stripped of any sensitive information. Do not share confidential information, personal information, or protected health information with this workforce. Do not use the Amazon Mechanical Turk workforce when you use Amazon Augmented AI in conjunction with other AWS HIPAA-eligible services, such as Amazon Textract and Amazon Rekognition.

Next Steps

If you are new to Amazon A2I and are integrating a human review workflow with a Amazon Rekognition or Amazon Textract task, we recommend that you start by creating a human review workflow using the console. For more information, see Create a Human Review Workflow (Console) (p. 107). If you are creating a human review workflow for a custom machine learning task, start by following the steps found in the Use Amazon Augmented AI with Custom Task Types (p. 112) topic.

You can also create a human review workflow using the Amazon A2I CreateFlowDefinition API. For more information, see Create a Human Review Workflow (API) (p. 108). First, you need to create a worker template to specify the worker UI. To learn how to create a worker template, for Amazon Rekognition, Amazon Textract, and custom labeling jobs, see Create a Worker UI (p. 113).
evaluating all form keys in the text, confidence is lower than your required threshold for any form key.

6. In the Create template section choose Build from a default template to create instructions for your workers using an Amazon Augmented AI default template for Amazon Rekognition and Amazon Textract task types, or choose Use your own template to import your own custom template. If you are using a custom task type, you need to have already created a template. To preview your instructions, choose See a sample worker task.

   - If you choose Build from a default template, create your instructions in Worker task design. See Creating Good Worker Instructions (p. 120) for additional information on creating effective instructions.
   - provide a Template name that is unique in the AWS Region you are in.
   - Use the Instructions section to provide detailed instructions on how to complete your task. To help workers achieve great accuracy, provide good and bad examples.
   - (Optional) Provide Additional instructions to provide your workers with additional information and instructions.
   - To learn how to create a custom template, see Create a Custom Worker Template (Console) (p. 113). After you've created a template, choose it from the Template drop-down menu and provide a Task description to briefly describe the task for your workers.

7. In Workers, choose a workforce type.

8. Choose Create.

After you've created your human loop workflow, it shows up in the console under Human review workflows. Choose your flow definition to see its ARN and configuration details.

Create a Human Review Workflow (API)

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Create a human review workflow using the Amazon Augmented AI API to incorporate a human data review for Amazon Rekognition, Amazon Textract, and custom tasks.

Important
This topic assumes that you have completed these Prerequisites (p. 106). Additionally, you need to create worker instructions, referred to as a worker template, using the Amazon SageMaker console. To learn how, see Create a Custom Worker Template (Console) (p. 113). Take note of the Amazon Resource Name (ARN) of your worker template, which is required input in Step 1.

First, you need to create a flow definition using the section called "CreateFlowDefinition" (p. 885), and then you incorporate that workflow into your task with StartHumanLoop.

Step 1: Create a FlowDefinition

When you create a flow definition using CreateFlowDefinition, the output is a FlowDefinitionArn. This is the ARN for your new human workflow definition. You need this ARN for Step 2. For an overview of the API used in the following procedure, and details about each attribute described below, see the section called "CreateFlowDefinition" (p. 885).

To create a flow definition

1. For FlowDefinitionName, enter a unique name for your flow definition.
2. For RoleArn, enter the role that you configured for access to Amazon SageMaker and your data sources.

3. For HumanLoopConfig, enter information about the workers and what they should see. This includes the following:
   - work team ARN
   - human task (worker template) UI ARN
   - number of tasks limit
   - task title
   - task description

4. (Optional) For HumanLoopActivationConfig, you can provide conditions that trigger a loop here. To learn how to create the input required for HumanLoopActivationConfig, see JSON Schema for Human Fallback Conditions in Amazon Augmented AI (p. 125). Note that if you are using a custom task type, this attribute will be disabled. For more information, see Use Amazon Augmented AI with Custom Task Types (p. 112).

5. For OutputConfig, indicate where the outputs of the human loop should be stored.

**Step 2: Start the Human Loop**

Use the following procedure to configure your human loop using StartHumanLoop. For a template of the API used below, see StartHumanLoop.

**To start the human loop**

1. For DataAttributes, please specify a set of ContentClassifiers related to the input provided to the StartHumanLoop action. Note that to use Amazon Mechanical Turk, you must set the FreeOfPersonallyIdentifiableInformation content classifier.
2. For FlowDefinitionArn, enter the ARN that was output from the previous step.
3. For HumanLoopInput, enter input data, output data, and configuration information in JSON format. This is also where you can configure inputs from compatible services such as Amazon Rekognition and Amazon Textract. For more information, see Using Amazon Augmented AI with Amazon Textract and Using Amazon Augmented AI with Amazon Rekognition.
4. For HumanLoopName, enter the unique name for the human loop.

When your human loop is initiated during the execution of your task, a HumanLoop resource is created and manages the resulting human review. The output from the human loop is a HumanLoopArn. This is the ARN for your new human loop. You will need this in Step 3. You can monitor the results of your human review using CloudWatch Events. For more information, see Use Amazon CloudWatch Events in Amazon Augmented AI (p. 123).

**Step 3: Review Your Output Data**

**To check your output data**

1. Check the results by calling DescribeHumanLoop. HumanLoopActivationResults contains information about the reason and outcome of activation of the loop.
2. Check the output data from your human loop in Amazon S3. The path to the data uses the following pattern.

   \[s3://customer-output-bucket-specified-in-flow-definition/flow-definition-name/YYYY/MM/DD/human-loop-name/HumanLoopCreationTimeStamp/output.json\]
If you want, you can integrate this structure with AWS Glue or Amazon Athena. For more information, see Managing Partitions for ETL Output in AWS Glue.

Use Task Types

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use Augmented AI to incorporate a human review into your workflow for Amazon Textract, Amazon Rekognition, or your own custom tasks. The topics in this section provide an overview of the specific task types supported through Amazon Augmented AI and examples of what workers might see when they log into their worker console. To learn how to create a human review workflow, see Get Started with Amazon Augmented AI (p. 106).

Topics
- Use Amazon Augmented AI with Amazon Textract (p. 110)
- Use Amazon Augmented AI with Amazon Rekognition (p. 111)
- Use Amazon Augmented AI with Custom Task Types (p. 112)

Use Amazon Augmented AI with Amazon Textract

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Textract enables you to add document text detection and analysis to your applications. It works with formatted text, and it can detect words and lines of words that are located close to each other. It can also analyze a document for items such as related text, tables, key-value pairs, and selection elements. You can add a human review loop into your textract inference jobs using Amazon Augmented AI. For example, if you want a human to review specific key-value pairs, or to review a document if inference confidence is below a certain threshold, you can create a trigger to start a human review of the data.

A2I Textract Worker Console Preview

When they’re assigned a review task in an Amazon Textract workflow, workers might see UI similar to the following:
Use Task Types

Integrate a Human Review into Amazon Textract

To integrate a human review into an Amazon Textract text detection and analysis job, you need to create a flow definition, and then use the Amazon Textract API to integrate that flow definition into your workflow. To learn how to create a flow definition using the Amazon SageMaker console or Augmented AI API, see the following topics:

- Create a Human Review Workflow (Console) (p. 107)
- Create a Human Review Workflow (API) (p. 108)

After you've created your flow definition, see Using Augmented AI with Amazon Textract to learn how to integrate your flow definition into your Amazon Textract task.

Use Amazon Augmented AI with Amazon Rekognition

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Rekognition enables you to add image and video analysis to your applications. It includes a simple API that can quickly analyze any image or video file that’s stored in Amazon S3. When using Amazon Rekognition for your image and video analysis, you can get a human review of unsafe content, such as explicit adult content or violent content using Amazon Augmented AI.

A2I Rekognition Worker Console Preview

When they’re assigned a review task in an Amazon Rekognition workflow, workers might see UI similar to the following:
You can customize this interface in the Amazon SageMaker console when you create your human review definition, or by creating and using a custom template. To learn more, see Create a Worker UI (p. 113).

Integrating a Human Review into Amazon Rekognition

To integrate a human review into an Amazon Rekognition, see the following topics:

- Create a Human Review Workflow (Console) (p. 107)
- Create a Human Review Workflow (API) (p. 108)

After you've created your flow definition, see Using Augmented AI with Amazon Rekognition to learn how to integrate your flow definition into your Amazon Rekognition task.

Use Amazon Augmented AI with Custom Task Types

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use Amazon Augmented AI to incorporate a human review into your custom Machine Learning workflow. To start, you will need to create a human review task template. Then, you will need to create a human review workflow using either the Amazon A2I API or Augmented AI section of the Amazon SageMaker console. These steps are documented throughout the Amazon A2I documentation. Follow the links below, in order, to integrate your human review workflow into your custom task.

1. Create a custom worker template and preview and test it using the Augmented AI section of the Amazon SageMaker console. You use this task template ARN when creating your flow definition using the Amazon A2I CreateFlowDefinition API.
   - To learn how to create a custom template, see Create Custom Templates (p. 114).
   - To learn how to preview and test your template in the Amazon SageMaker console, see Create a Custom Worker Template (Console) (p. 113).
2. Create a human review workflow using the Amazon A2I CreateFlowDefinition API. To learn how, see Create a Human Review Workflow (API) (p. 108).
### Create a Worker UI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can create a UI for your workers with detailed instructions on how to complete your task directly in the Amazon SageMaker console or by using a custom template. You can preview your worker UI using the Augmented AI Worker template preview tool in the Amazon SageMaker console. If you are using a custom task type, you will need to create a custom template using the instructions found in [Create Custom Templates](#) and then use the Amazon SageMaker console to preview and test your template.

**Topics**
- [Create a Custom Worker Template (Console)](#)
- Create Custom Templates (p. 114)
- Creating Good Worker Instructions (p. 120)

### Create a Custom Worker Template (Console)

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use a worker template to customize the interface and instructions that your workers see when working on your tasks. Using the procedures in this topic, you can create a custom worker task template in the Amazon SageMaker console. A starter template is provided for Amazon Textract and Amazon Rekognition tasks. If you are using a custom task type, you need to create a custom template using the instructions found in Create Custom Templates (p. 114), and then use the procedure below to preview and test your template.

**To create a worker task template in the Amazon SageMaker console**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/
2. In Amazon Augmented AI in the left navigation pane, choose Worker task templates.
3. Choose Create template.
4. In Template name, enter a unique name.
5. (Optional) Enter an IAM role that grants A2I the permissions necessary to call services on your behalf.
6. In Template type, choose a template type from the drop-down menu. If you are creating a template for a Textract-form extraction or Rekognition-image moderation task, choose the appropriate option.
7. Enter your custom template elements as follows:
• If you selected the Amazon Textract or Amazon Rekognition task template, the Template editor autopopulates with a default template that you can customize.
• If you are using a custom template, enter your predefined template in the editor.

8. Choose See preview to preview the interface and instructions that workers will see. This is an interactive preview. After you complete the sample task and choose Submit, you see the resulting output from the task that you just performed.

9. When you're satisfied with your template, choose Create.

After you've created your template, you can select that template when you create a human review workflow in the console. Your template also appears in the Amazon Augmented AI section of the Amazon SageMaker console under Worker task templates. Choose your template to view its ARN. You can use this ARN to specify your worker UI when you create a flow definition using the API.

Create Custom Templates

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Crowd HTML Elements are web components that provide a number of task widgets and design elements that you can tailor to the question you want to ask. You can use these crowd elements to create a custom worker template and integrate this template with an Amazon Augmented AI human review workflow to customize the worker console and instructions.

For a list of all HTML crowd elements available to Augmented AI users, see HTML Elements Reference (p. 142). To see examples of templates, go to the AWS Github repository, which contains over 60 sample custom task templates.

Develop Templates Locally

While you need to be in the console to test how your template process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding the following code to the top of your HTML file:

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you want to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this won't parse your variables. You may want to replace them with sample content while developing locally.

Use External Assets

Amazon Augmented AI custom templates enable you to embed external scripts and style sheets.

Example

```html
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.
For example, the MIME and encoding types for remote scripts is `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets is `text/css;CHARSET=UTF-8`.

**Track Your Variables**

In the process of building the sample below, there is a step where you add variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you need to make sure you're aware of the variables it already uses.

For example, for a custom template that integrates an Augmented AI human review loop with an Amazon Textract text review task uses `{{ task.input.selectedAiServiceResponse.blocks }}` for initial-value input data. For an Augmented AI integration with Amazon Rekognition, `{{ task.input.selectedAiServiceResponse.moderationLabels }}` is used.

For a custom task type, you need to determine the input parameter for your task type. Use `{{ task.input.customInputValuesForStartHumanLoop }}` where you specify `customInputValuesForStartHumanLoop`.

**Custom Template Example for Amazon Textract**

All custom templates begin and end with the `<crowd-form> </crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them.

For a Textract-analysis task, use the `<crowd-textract-document-analysis>` element. It uses the following attributes:

- `src` – The URL of the image file to be annotated.
- `initialValue` – Sets initial values for attributes found in the worker UI.
- `blockTypes` (required) – Determines the kind of analysis that the workers can do. Only `KEY_VALUE_SET` is currently supported.
- `keys` (required) – Specifies new keys and the associated text value the worker can add.
- `no-key-edit` (required) – Prevents the workers from editing the keys of annotations passed through `initialValue`.
- `no-geometry-edit` – Prevents workers from editing the polygons of annotations passed through `initialValue`.

As children of the `<crowd-textract-document-analysis>` element, you must have two Regions. You can use arbitrary HTML and CSS elements in these Regions.

- `<full-instructions>` – Instructions that are available from the "View full instructions" link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- `<short-instructions>` – A brief description of the task that appears in the tool's sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

An example of this tool would look like the following.

**Example**

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_arn %}http://s3.amazonaws.com/{{ task.input.aiServiceRequest.document.s3Object.bucket }}/{{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}
```
Create a Worker UI

- View the key-value pairs listed on the right and correct them if they don't match the following document.
- If it is a valid key-value pair, review the content for the value. If the content is incorrect, correct it.
- The text of the value is incorrect, correct it.
- A wrong value is identified, correct it.
- If it is not a valid key-value relationship, choose No.
- If you can’t find the key in the document, choose Key not found.
- If the content of a field is empty, choose Value is blank.
- Key and value are often displayed next or below to each other.
- Key and value displayed in one line.
- Key and value displayed in two lines.
- If the content of the value has multiple lines, enter all the text without line break.

Examples:
- Key and value displayed in one line.
- Key and value displayed in two lines.
Custom Template Example for Amazon Rekognition

All custom templates begin and end with the `<crowd-form> </crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them. For an Amazon Rekognition custom task template, use the `<crowd-rekognition-detect-moderation-labels>` element. This element supports the following attributes:

- **categories** – An array of strings or an array of objects where each object has a `name` field.
  - If the categories come in as objects, the following applies:
    - The displayed categories are the value of the `name` field.
    - The returned answer contains the **full** objects of any selected categories.
  - If the categories come in as strings, the following applies:
    - The returned answer is an array of all the strings that were selected.
- **exclusion-category** – By setting this attribute, you create a button underneath the categories in the UI. When a user chooses the button, all categories are deselected and disabled. If you choose the button again, you re-enable users to choose categories. If you submit after choose the button, it returns an empty array.

As children of the `<crowd-textract-document-analysis>` element, you must have three Regions.

- **<full-instructions>** – Instructions that are available from the View full instructions link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- **<short-instructions>** – Brief description of the task that appears in the tool’s sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

An example of a template using these elements would look like the following.

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_arn %}http://s3.amazonaws.com/{{ task.input.aiServiceRequest.image.s3Object.bucket }}/{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
categories='[
    {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
    {name: "{ label.name }",
     parentName: "{ label.parentName }",
    },
    {% endfor %}
  ]'
src="{{ s3_arn | grant_read_access }}"
header="Review the image and choose all applicable categories."
>
  <short-instructions header="Instructions"><p class='instructions'>Review the image and choose all applicable categories. If no categories apply, choose None.</p></short-instructions>
</crowd-form>
```
Add Automation with Liquid

The custom template system uses Liquid for automation. It is an open source inline markup language. For more information and documentation, see the Liquid homepage.
In Liquid, the text between single curly braces and percent symbols is an instruction or tag that creates control flow. Text between double curly braces is a variable or object that outputs its value.

Variable Filters

In addition to the standard Liquid filters and actions, Augmented AI offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, and then specifying a filter name. Filters can be chained in the following format:

Example

```
{{ <content> | <filter> | <filter> }}
```

Autoescape and Explicit Escape

By default, inputs are HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the escape filter to make it more obvious to someone reading the source of your template that the escaping is being done.

escape_once

escape_once ensures that if you've already escaped your code, it doesn't get re-escaped again. For example, to ensure that & doesn't become &amp;.&

skip_autoescape

skip_autoescape is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

Use skip_autoescape sparingly

The best practice in templates is to avoid passing in functional code or markup with skip_autoescape unless you are absolutely sure you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a cross-site scripting attack.

to_json

to_json encodes what you feed it to JavaScript Object Notation (JSON). If you feed it an object, it serializes it.

grant_read_access

grant_read_access takes an Amazon S3 URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible.

Example of the filters

Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?
}}
explicit escape: {{ "Have you read 'James & the Giant Peach'" | escape }}
explicit escape_once: {{ "Have you read 'James &amp; the Giant Peach'" | escape_once }}
skip_autoescape: {{ "Have you read 'James & the Giant Peach'" | skip_autoescape }}
to_json: {{ jsObject | to_json }}
grant_read_access: {{ "s3:///mybucket/myphoto.png" | grant_read_access }}
```
Example

Output

auto-escape: Have you read &amp;#39;James &amp;amp; the Giant Peach&amp;#39;?
explicit escape: Have you read &amp;#39;James &amp;amp; the Giant Peach&amp;#39;?
explicit escape_once: Have you read &amp;#39;James &amp;amp; the Giant Peach&amp;#39;?
skip autoscape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?access token and other params>

Example of an automated classification template.

To automate this simple text classification sample, include the Liquid tag {{ task.input.source }}. This example uses the crowd-classifier (p. 151) element.

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories=['positive', 'negative', 'neutral', 'cannot determine']
    header="Which term best describes this tweet?"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seem to match, choose "other."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>

Creating Good Worker Instructions

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Creating good instructions for your human review jobs improves your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console when creating a human review workflow, or you can use the console to create a custom worker template and include your instructions in this template. The instructions are shown to the worker on the UI page where they complete their labeling task.

Create Good Worker Instructions

There are three kinds of instructions in the Amazon Augmented AI console:
Add Example Images to Your Instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions, do the following:

1. Place the cursor where the image should go in the instructions editor.
2. Choose the image icon in the editor toolbar.
3. Enter the URL of your image.

If your instruction image is in an S3 bucket that isn't publicly accessible, do the following:

- For the image URL, enter: `{ { 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access } }`.

This renders the image URL with a short-lived, one-time access code that's appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview. See grant_read_access (p. 119) for more information about the grant_read_access element.

Permissions and Security in Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

When using Amazon Augmented AI, you create resources, such as workflows and templates. Certain resources require permissions to access other AWS services, for example, Amazon Simple Storage Service (Amazon S3). Both the human review workflow and the worker task templates require these permissions.

Creating an IAM Role that grants Augmented AI Permissions to Call Other Services On Your Behalf

When creating a human review workflow (or FlowDefinition), you need an AWS Identity and Access Management (IAM) role that grants Augmented AI permissions to call other AWS services on your behalf.

You pass this role when you call the CreateFlowDefinition API. The role needs permissions to access Amazon S3, both for reading objects that will be rendered in a human task UI and for writing the results of the human review. This role also needs to trust the sagemaker.amazonaws.com service principal. This allows the role to be assumed by Augmented AI.

For example, if the images, documents, and so on that you are sending for human review are stored in an S3 bucket named my_input_bucket, and if you want the human reviews to be stored in a bucket
named `my_output_bucket`, you would attach the following IAM policy to the role that is passed to the `CreateFlowDefinition` API.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["s3:GetObject"],
         "Resource": ["arn:aws:s3:::my_input_bucket/*"]
      },
      {
         "Effect": "Allow",
         "Action": ["s3:PutObject"],
         "Resource": ["arn:aws:s3:::my_output_bucket/*"]
      }
   ]
}
```

In addition, the IAM role must have the following trust policy.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowSageMakerToAssumeRole",
         "Effect": "Allow",
         "Principal": {
            "Service": "sagemaker.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

**Creating an IAM Role that Can Invoke Augmented AI Operations**

To use Augmented AI with Amazon Rekognition, Amazon Textract, or the Augmented AI runtime API, you must have an IAM role that has permissions to invoke Augmented AI operations. Attach the AmazonAugmentedAIFullAccess policy to a new or existing IAM user.

**To create the required IAM role**

2. Choose an existing user or create a new user.
   
   For more information, see Creating an IAM User in Your AWS Account in the AWS Identity and Access Management User Guide.
3. Choose the user. This displays the summary page for the user.
4. Choose Add permissions.
5. Choose Attach existing policies directly.
6. Choose AmazonAugmentedAIFullAccess.

If you don’t see the name on the first page of the list, filter the policies or search for it.

7. Choose Next: Review.

8. Choose Add permissions.

For more information, see Adding and Removing IAM Identity Permissions in the AWS Identity and Access Management User Guide.

**HumanTaskUi Template Preview Permissions**

To create a HumanTaskUi template, you need an IAM role with permissions to read Amazon S3 objects that get rendered on your user interface. This role enables you to preview your template. It is passed to the RenderUiTemplate operation. If you already have an IAM role, you can attach the following example policy to it to enable previewing a template.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["s3:GetObject"],
         "Resource": ["arn:aws:s3:::my_input_bucket/*"]
      }
   ]
}
```

**Additional Resources for Amazon SageMaker and Augmented AI**

- the section called “Control Access to Amazon SageMaker Resources by Using Tags” (p. 754).
- the section called “Amazon SageMaker Identity-Based Policies” (p. 732)
- the section called “Control Creation of Amazon SageMaker Resources with Condition Keys” (p. 745)
- the section called “Amazon SageMaker API Permissions Reference” (p. 771)
- Security (p. 724)

**Use Amazon CloudWatch Events in Amazon Augmented AI**

Amazon Augmented AI uses Amazon CloudWatch Events to alert you when a human review loop changes status. When a review loop changes to the Completed, Failed, or Stopped status, Augmented AI sends a response to CloudWatch Events similar to the following:

```json
{
   "version":"0",
   "id":"12345678-1111-2222-3333-12345EXAMPLE",
   "detail-type":"Amazon Augmented AI HumanLoop Status Change",
   "source":"aws.sagemaker",
   "account":"111111111111",
   "time":"2019-11-14T17:49:25Z",
}
```
The details in the JSON output include the following:

**creationTime**

The timestamp when Augmented AI created the human loop.

**failureCode**

A failure code denoting a specific type of failure.

**failureReason**

The reason why a human loop has failed. The failure reason is only returned when the human review loop status is failed.

**flowDefinitionArn**

The Amazon Resource Name (ARN) of the flow definition, or human review workflow.

**humanLoopArn**

The Amazon Resource Name (ARN) of the human loop.

**humanLoopName**

The name of the human loop.

**humanLoopOutput**

An object containing information about the output of the human loop.

**outputS3Uri**

The location of the Amazon S3 object where Augmented AI stores your human loop output.

**humanLoopStatus**

The status of the human loop.

### Use Human Review Output

After you receive human review results, you can analyze the results and compare them to machine learning predictions. The JSON that is stored in the Amazon S3 bucket contains both the machine learning predictions and the human review results.
More Information

the section called “React to Amazon SageMaker Job Status Changes with CloudWatch Events” (p. 723)

Use APIs in Amazon Augmented AI (Preview)

You can create a human review workflow or a worker task template programmatically. The APIs you use depend on whether you are creating a Amazon Rekognition, Amazon Textract, or custom task type. This topic provides links to API reference documentation for each task type and programming task.

The following APIs can be used with Augmented AI:

**Amazon Augmented AI**

Use the Augmented AI API to start, stop, and delete human review loops. You can also list all human review loops and return information about human review loops in your account.

Learn more about human review loop APIs in the Amazon Augmented AI Runtime API Reference.

**Amazon Rekognition**

Use the `HumanLoopConfig` parameter of the `DetectModerationLabels` API to trigger a human review workflow using Amazon Rekognition.

**Amazon SageMaker**

Use the Amazon SageMaker API to create a `FlowDefinition`, also known as a human review workflow. You can also create a `HumanTaskUi`, or worker task template.

For more information, see the the section called “CreateFlowDefinition” (p. 885) or the the section called “CreateHumanTaskUi” (p. 888) API documentation.

**Amazon Textract**

Use the `HumanLoopConfig` parameter of the `AnalyzeDocument` API to trigger a human review workflow using Amazon Textract.

Programmatic Walkthroughs

The following walkthroughs and tutorials provide example code and step-by-step instructions for creating human review workflows and worker task templates programmatically.

- the section called “Create a Human Review Workflow (API)” (p. 108)
- Using Amazon Augmented AI with Amazon Rekognition in the Amazon Rekognition Developer Guide
- Using Amazon Augmented AI with Amazon Textract AnalyzeDocument in the Amazon Textract Developer Guide

JSON Schema for Human Fallback Conditions in Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.
The HumanTaskActivationConditions is an input parameter of the section called "CreateFlowDefinition" (p. 885) API. This parameter is a JSON formatted string. The JSON models the conditions under which a HumanLoop will be created, when those conditions are evaluated against the response from an integrating AI service API (such as Rekognition_DETECT_MODERATION_LABELS or Textract_ANALYZE_DOCUMENT).

The schema for the JSON can be found below. At the top level, the HumanTaskActivationConditions has a JSON array, Conditions. Each member of this array is an independent condition that, if evaluated to true, will result in Amazon A2I creating a HumanLoop. Each such independent condition can be a primitive condition or a complex condition. A simple condition has the following attributes:

- **ConditionType**: This attribute identifies the type of condition. Each AWS AI service API that integrates with Amazon A2I defines its own set of allowed ConditionTypes.
  - **Rekognition_DETECT_MODERATIONLABELS** – This API supports the ModerationLabelConfidenceCheck ConditionType value. No other ConditionTypes are supported.
  - **Textract_ANALYZEDOCUMENT** – This API supports the ImportantFormKeyConfidenceCheck ConditionType value. No other ConditionTypes are supported.
  - **ConditionParameters**: This is a JSON object that parameterizes the condition. The set of allowed attributes of this object is dependent on the value of the ConditionType. Each ConditionType defines its own set of ConditionParameters.

A member of the Conditions array can model a complex condition. This is accomplished by logically connecting primitive conditions using the And, Or, and Not logical operators, nesting the underlying primitive conditions. Up to two levels of nesting are supported.

```json
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "definitions": {
      "Condition": {
         "type": "object",
         "properties": {
            "ConditionType": {
               "type": "string"
            },
            "ConditionParameters": {
               "type": "object"
            }
         },
         "required": ["ConditionType"]
      },
      "OrConditionArray": {
         "type": "object",
         "properties": {
            "Or": {
               "type": "array",
               "minItems": 2,
               "items": {
                  "$ref": "#/definitions/ComplexCondition"
               }
            }
         }
      },
      "AndConditionArray": {
         "type": "object",
         "properties": {
            "And": {
               "type": "array",
               "items": {
                  "$ref": "#/definitions/ComplexCondition"
               }
            }
         }
      }
   }
}
```
"type": "array",
"minItems": 2,
"items": {
    "$ref": "#/definitions/ComplexCondition"
}
}

"NotCondition": {
    "type": "object",
    "properties": {
        "Not": {
            "$ref": "#/definitions/ComplexCondition"
        }
    }
}

"ComplexCondition": {
    "anyOf": [
        {
            "$ref": "#/definitions/Condition"
        },
        {
            "$ref": "#/definitions/OrConditionArray"
        },
        {
            "$ref": "#/definitions/AndConditionArray"
        },
        {
            "$ref": "#/definitions/NotCondition"
        }
    ]
}

"type": "object",
"properties": {
    "Conditions": {
        "type": "array",
        "items": {
            "$ref": "#/definitions/ComplexCondition"
        }
    }
}

Use Human Fallback Conditions JSON Schema with Amazon Textract

Textract AnalyzeDocument supports the ImportantFormKeyConfidenceCheck ConditionType.

For this ConditionType, the following ConditionParameters are supported:

- **ImportantFormKey** – A string representing a key in a key-value block detected by Textract that needs to be reviewed by human workers. If the value of this parameter is the special catch-all value (*), then all keys are considered to be matched to the condition. This can be used to model the case where any key-value pair satisfying certain confidence thresholds would need human review.
- **ImportantFormKeyAliases** – An array that represents alternate spellings or logical equivalents for the important form key.
- **KeyValueBlockConfidenceEquals**
- **KeyValueBlockConfidenceLessThan**
- **KeyValueBlockConfidenceLessThanEquals**
Following is an example of a `HumanLoopActivationConditions` JSON that triggers a HumanLoop if any one of the following three conditions is met:

- **Textract AnalyzeDocument API** returns a key-value pair whose key is one of `Employee Name`, `Name`, or `EmployeeName`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.
- **Textract AnalyzeDocument API** returns a key-value pair whose key is one of `Pay Date`, `PayDate`, `DateOfPay`, or `pay-date`, with the confidence of the key-value block being less than 65 and the confidences of each of the Word blocks making up the key and value being less than 85.
- **Textract AnalyzeDocument API** returns a key-value pair whose key is one of `Gross Pay`, `GrossPay`, or `GrossAmount`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.

While only one condition needs to evaluate to true to trigger a HumanLoop, A2I actually evaluates all conditions. The human reviewers are asked to review the important form keys for all the conditions that evaluated to true.

**Example 1: Specifically defined important form keys**

```json
{
  "Conditions": [
    {
      "ConditionType": "ImportantFormKeyConfidenceCheck",
      "ConditionParameters": {
        "ImportantFormKey": "Employee Name",
        "ImportantFormKeyAliases": ["Name", "EmployeeName"],
        "KeyValueBlockConfidenceLessThan": 60,
        "WordBlockConfidenceLessThan": 85
      }
    },
    {
      "ConditionType": "ImportantFormKeyConfidenceCheck",
      "ConditionParameters": {
        "ImportantFormKey": "Pay Date",
        "ImportantFormKeyAliases": ["PayDate", "DateOfPay", "pay-date"],
        "KeyValueBlockConfidenceLessThan": 65,
        "WordBlockConfidenceLessThan": 85
      }
    },
    {
      "ConditionType": "ImportantFormKeyConfidenceCheck",
      "ConditionParameters": {
```
Example 2: All detected key-value pairs trigger HumanLoop

In the below example if there is any key-value pair detected by Amazon Textract whose confidence for the key-value block is less than 60 and for any underlying word blocks is less than 90, a HumanLoop is created. The human reviewers are asked to review all the form key-value pairs that matched the confidence value comparisons.

```
{
  "Conditions": [
    {
      "ConditionType": "ImportantFormKeyConfidenceCheck",
      "ConditionParameters": {
        "ImportantFormKey": "*",
        "KeyValueBlockConfidenceLessThan": 60,
        "WordBlockConfidenceLessThan": 90
      }
    }
  ]
}
```

Use Human Fallback Conditions JSON Schema with Amazon Rekognition

Rekognition DetectModerationLabels API supports the ModerationLabelConfidenceCheck ConditionType. For this ConditionType, the following ConditionParameters are supported:

- **ModerationLabelName** – The exact (case sensitive) name of a ModerationLabel detected by the Amazon Rekognition DetectModerationLabels API. You can specify the special catch-all value (*) to denote any moderation label.
- ConfidenceEquals
- ConfidenceLessThan
- ConfidenceLessThanEquals
- ConfidenceGreaterThan
- ConfidenceGreaterThanEquals

Example 1: Explicit moderation labels

Following is an example of HumanLoopActivationConditions that trigger a HumanLoop when any of the following two conditions are met:

1. Amazon Rekognition detects the Graphic Male Nudity moderation label with a confidence between 90 and 99.
2. Amazon Rekognition detects the Graphic Female Nudity moderation label with a confidence between 80 and 99.
Note the use of the Or and And logical operators to model the above logic.

While any one of the two conditions under the Or operator need to evaluate to true for a HumanLoop to be created, Amazon Augmented AI actually evaluates all conditions. Human reviewers are asked to review the moderation labels for all the conditions that evaluated to true.

```
{
  "Conditions": [{
    "Or": [{
      "And": [{
        "ConditionType": "ModerationLabelConfidenceCheck",
        "ConditionParameters": {
          "ModerationLabelName": "Graphic Male Nudity",
          "ConfidenceLessThanOrEqual": 99
        }
      },
      { "ConditionType": "ModerationLabelConfidenceCheck",
        "ConditionParameters": {
          "ModerationLabelName": "Graphic Male Nudity",
          "ConfidenceGreaterThanOrEqual": 90
        }
      }
    },
    { "And": [{
        "ConditionType": "ModerationLabelConfidenceCheck",
        "ConditionParameters": {
          "ModerationLabelName": "Graphic Female Nudity",
          "ConfidenceLessThanOrEqual": 99
        }
      },
      { "ConditionType": "ModerationLabelConfidenceCheck",
        "ConditionParameters": {
          "ModerationLabelName": "Graphic Female Nudity",
          "ConfidenceGreaterThanOrEqual": 80
        }
      }
    }]
  }]
}
```

**Example 2: Any moderation labels**

In the following example, if any moderation label is detected with a confidence greater than 75, a HumanLoop is triggered. Human reviewers are asked to review all moderation labels that met the confidence value comparison.

```
{
  "Conditions": [
    {
      "ConditionType": "ModerationLabelConfidenceCheck",
      "ConditionParameters": {
        "ModerationLabelName": "*",
        "ConfidenceGreaterThanOrEqual": 75
      }
    }
  ]
}
```
Create and Manage Workforces

A workforce is the group of workers that you have selected to label your dataset. You can choose either the Amazon Mechanical Turk workforce, a vendor-managed workforce, or you can create your own private workforce to label or review your dataset. Whichever workforce type you choose, Amazon SageMaker takes care of sending tasks to workers.

When you use a private workforce, you also create work teams, a group of workers from your workforce that are assigned to specific jobs—Amazon SageMaker Ground Truth labeling jobs or Amazon Augmented AI human review tasks. You can have multiple work teams and can assign one or more work teams to each job.

Ground Truth and Amazon Augmented AI use Amazon Cognito to manage your private workforce and work teams. For more information about the permissions required to manage your workforce this way, see Permissions Required to Use the Amazon SageMaker Ground Truth Console (p. 737).

Topics
- Using the Amazon Mechanical Turk Workforce (p. 131)
- Managing Vendor Workforces (p. 132)
- Use a Private Workforce (p. 133)

Using the Amazon Mechanical Turk Workforce

The Amazon Mechanical Turk workforce provides the most workers for your Amazon Augmented AI task review and Amazon SageMaker Ground Truth labeling job.

You can use the console to choose the Amazon Mechanical Turk workforce for your Amazon SageMaker Ground Truth labeling job or Amazon Augmented AI human review workflow, or you can provide the Amazon Resource Name (ARN) for the Amazon Mechanical Turk workforce when you use the Amazon A2I CreateLabelingJob (p. 897) operation.

Any Amazon Mechanical Turk workforce billing is handled as part of your Ground Truth or Amazon Augmented AI billing. You do not need to create a separate Mechanical Turk account to use the Amazon Mechanical Turk workforce.

The ARN for the Amazon Mechanical Turk workforce is:


The Amazon Mechanical Turk workforce is a world-wide resource. Workers are available 24 hours a day, 7 days a week. You typically get the fastest turn-around for your human review tasks and labeling jobs when you use the Amazon Mechanical Turk workforce.

Adjust the number of workers that annotate each data object based on the complexity of the job and the quality that you need. Amazon SageMaker Ground Truth uses annotation consolidation to improve the quality of the labels. More workers can make a difference in the quality of the labels for more complex labeling jobs, but might not make a difference for simpler jobs. For more information, see Annotation Consolidation (p. 63). Annotation consolidation is not supported for Amazon Augmented AI human review workflows.
Important
You should not share confidential information, personal information or protected health information with this workforce. For avoidance of doubt, you should not use the Amazon Mechanical Turk workforce when you use Amazon A2I in conjunction with AWS HIPAA-eligible services, such as Amazon Textract and Amazon Rekognition for workloads containing protected health information.

To choose the Amazon Mechanical Turk workforce when you are creating a labeling job or human review workflow using the console, do the following during the Select workers and configure tool step:

To use the Amazon Mechanical Turk workforce
1. Choose Public from Worker types.
2. Choose The dataset does not contain adult content if your dataset doesn't contain potentially offensive content. This enables workers to opt out if they don't want to work with it.
3. Acknowledge that your data will be viewed by the Amazon Mechanical Turk workforce and that all personally identifiable information (PII) has been removed.
4. Choose Additional configuration to set optional parameters.
5. Optional. Enable automated data labeling to have Ground Truth automatically label some of your dataset. For more information, see Using Automated Data Labeling (p. 65). Automated data labeling is not available for Amazon Augmented AI.
6. Optional. Set the number of workers that should see each object in your dataset. Using more workers can increase the quality of your labels but also increases the cost.

Your labeling job or human review task will now be sent to the Amazon Mechanical Turk workforce. You can use the console to continue configuring your labeling job.

Managing Vendor Workforces
You can use a vendor-managed workforce to label your data using Amazon SageMaker Ground Truth (Ground Truth) and Amazon Augmented AI (Amazon A2I). Vendors have extensive experience in providing data labeling services for the purpose of machine learning. Vendor workforces for these two services must be created and managed separately through the Amazon SageMaker console.

Vendors make their services available via the AWS Marketplace. You can find details of the vendor's services on their detail page, such as the number of workers and the hours that they work. You can use these details to make estimates of how much the labeling job will cost and the amount of time that you can expect the job to take. Once you have chosen a vendor you subscribe to their services using the AWS Marketplace.

A subscription is an agreement between you and the vendor. The agreement spells out the details of the agreement, such as price, schedule, or refund policy. You work directly with the vendor if there are any issues with your labeling job.

You can subscribe to any number of vendors to meet your data annotation needs. When you create a labeling job or human review workflow you can specify that the job be routed to a specific vendor.

Before you send sensitive data to a vendor, check the vendor's security practices on their detail page and review the end user license agreement (EULA) that is part of your subscription agreement.

You must use the console to subscribe to a vendor workforce. Once you have a subscription, you can use the ListSubscribedWorkteams (p. 1168) operation to list your subscribed vendors.

To subscribe to a vendor workforce
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose the appropriate page in the Amazon SageMaker console.
• For Ground Truth labeling jobs, choose **Labeling workforces**, choose **Vendor**, and then choose **Find data labeling services**.
• For Amazon A2I human review workflows, choose **Human review workforces**, choose **Vendor**, and then choose **Find human review services**.

3. The console opens the AWS Marketplace with:
   • data labeling services category selected for Ground Truth
   • human review services category selected for Amazon A2I
   
   Here you see a list of the vendor services available for this service.

4. Choose a vendor. The AWS Marketplace shows detailed information about the data labeling or human review service. Use this information to determine if the vendor meets your requirements for your task.

5. If the vendor meets your requirements, choose **Continue to subscribe**.

6. Review the details of the subscription. If you agree to the terms, choose **Subscribe** to complete your subscription to the service.

**Use a Private Workforce**

A **private workforce** is a group of workers that you choose. These can be employees of your company or a group of subject matter experts from your industry. For example, if the task is to label medical images, you could create a private workforce of people knowledgeable about the images in question.

Each AWS account has access to a single private workforce per region, and the owner has the ability to create multiple **private work teams** within that workforce. A single private work team is used to complete a labeling job or human review task, or a **job**. You can assign each work team to a separate job or use a single team for multiple jobs. A single worker can be in more than one work team.

**Topics**

• Create a Private Workforce (p. 133)
• Manage a Private Workforce (p. 136)
• Create and manage Amazon SNS topics for your work teams (p. 141)

**Create a Private Workforce**

There are three ways that you can create a private workforce:

• Create a new workforce while you are creating your labeling job. Do this in the Amazon SageMaker console in the Ground Truth section.
• Create a new workforce before you create your labeling job. Do this in the Amazon SageMaker console in the Ground Truth section.
• Import an existing workforce after creating a user pool in the Amazon Cognito console.

Once you create a private workforce, that workforce and all work teams and workers associated with it will be available to use for all Ground Truth labeling job tasks and Amazon Augmented AI human review workflows tasks. To use a private workforce for an Amazon Augmented AI human review workflow, you must first create the workforce using one of the three methods described above.

**Topics**

• Create a Private Workforce (Console) (p. 134)
• Create a Private Workforce (Amazon Cognito Console) (p. 135)
Create a Private Workforce (Console)

You can create a private workforce in the Amazon SageMaker console in one of two ways:

- When creating a labeling job in the **Labeling jobs** page of the Amazon SageMaker Ground Truth section
- Using the **Labeling workforces** page of the Amazon SageMaker Ground Truth section

Both of these methods also create a default work team containing all of the members of the workforce. This private workforce will be available to use for both Ground Truth and Amazon Augmented AI jobs.

Create a Workforce When Creating a Labeling Job

If you haven't created a private workforce when you create your labeling job, you are prompted to create one.

To create a workforce while creating a labeling job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Labeling jobs** and fill in all required fields. For instructions on how to start a labeling job, see Getting started (p. 50). Choose **Next**.
3. Choose **Private** for the workforce type.
4. In the **Workers** section, Enter:
   a. The **Team name**.
   b. Email addresses for up to 100 workforce members. Email addresses are case sensitive. Your workers must log in using the same case used when the address was initially entered. You can add additional workforce members after the job has been created.
   c. The name of your organization. Amazon SageMaker uses this to customize the email sent to the workers.
   d. A contact email address for workers to report issues related to the task.

When you create the labeling job, an email is sent to each worker inviting them to join the workforce. After creating the workforce, you can add, delete, and disable workers using the Amazon SageMaker console or the Amazon Cognito console.

Create a Workforce Using the Labeling Workforces Page

To create and manage your private workforce, you can use the **Labeling workforces** page. When following the instructions below, you will have the option to create a private workforce by entering worker emails or importing a pre-existing workforce from an Amazon Cognito user pool. To import a workforce, see Create a Private Workforce (Amazon Cognito Console) (p. 135).

To create a private workforce (worker emails)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**, then choose **Create private team**.
4. Choose **Invite new workers by email**.
5. Paste or type a list of up to 50 email addresses, separated by commas, into the email addresses box.
6. Enter an organization name and contact email.
7. Enter an organization name and contact email.
8. Optionally choose an SNS topic to which to subscribe the team so workers are notified by email when new labeling jobs become available.
9. Click the **Create private team** button.

After you import your private workforce, refresh the page. On the **Private workforce summary** page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce.

**Note**
If you delete all of your private work teams, you have to repeat this process to use a private workforce in that region.

**Create a Private Workforce (Amazon Cognito Console)**

Amazon Cognito is used to define and manage your private workforce and your work teams. It is a service that you can use to create identities for your workers and authenticate these identities with identity providers. A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool.

Example identity providers supported by Amazon Cognito:

- Social sign-in providers such as Facebook and Google
- OpenID Connect (OIDC) providers
- Security Assertion Markup Language (SAML) providers such as Active Directory
- The Amazon Cognito built-in identity provider

For more information, see [What Is Amazon Cognito?](#)

To create a private workforce using Amazon Cognito, you must have an existing Amazon Cognito user pool containing at least one user group. See [Tutorial: Creating a User Pool](#) to learn how to create a user pool. See [Adding Groups to a User Pool](#) to learn how to add a user group to a pool.

Once your user pool has been created, follow the steps below to create a private workforce by importing that user pool into Amazon SageMaker.

**To create a private workforce by importing a Amazon Cognito user pool**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**.
4. Choose **Create private team**. This creates a private workforce and a work team.
5. Choose **Import workers from existing Amazon Cognito user groups**.
6. Choose a user pool that you have created. User pools require a domain and an existing user group. If you get an error that the domain is missing, set it in the **Domain name** options on the **App integration** page of the Amazon Cognito console for your group.
7. Choose an app client. We recommend using a client generated by Amazon SageMaker.
8. Choose a user group from your pool to import its members.
9. Optionally choose an Amazon Simple Notification Service (Amazon SNS) topic to which to subscribe the team so that workers are notified by email when new labeling jobs become available.
10. Choose **Create private team**.

**Important**
After you create a workforce using a Amazon Cognito user pool, it should not be deleted without first deleting all work teams associated with that pool in the Amazon SageMaker console.
After you import your private workforce, refresh the page to see the **Private workforce summary** page. On this page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce. This workforce will now be available to use in both Amazon Augmented AI and Amazon SageMaker Ground Truth for human review tasks and data labeling jobs respectively.

### Manage a Private Workforce

After you have create a private workforce, you can manage work teams using either the Amazon SageMaker or Amazon Cognito console. If you add workers to a workforce using the Amazon Cognito console, you must use the same console to remove the worker from the workforce.

**Note**

Your private workforce is shared between Amazon SageMaker Ground Truth and Amazon Augmented AI. To manage private work teams and workers used by Amazon Augmented AI, use the Ground Truth section of the Amazon SageMaker console or the Amazon Cognito user pool used to create your shared private workforce.

**Topics**

- Manage a Private Workforce (Console) (p. 136)
- Manage a Private Workforce (Amazon Cognito Console) (p. 138)
- Track Worker Performance (p. 140)

### Manage a Private Workforce (Console)

You can use the Amazon SageMaker console to create and manage the work teams and individual workers that make up a private workforce.

Use a work team to assign members of your private workforce to a **job** - a labeling or human review job. When you create your workforce using the Amazon SageMaker console, there is a work team called **Everyone-in-private-workforce** that you can use if you want to assign your entire workforce to a job. Because an imported Amazon Cognito user pool may contain members that you don't want to include in your work teams, a similar work team is not created for Amazon Cognito user pools.

You have two choices to create a new work team:

- You can create a work team in the Amazon SageMaker console and add members from your workforce to the team.
- You can create a user group by using the Amazon Cognito console and then create a work team by importing the user group. You can import more than one user group into each work team. You manage the members of the work team by updating the user group in the Amazon Cognito console. See Manage a Private Workforce (Amazon Cognito Console) (p. 138) for more information.

### Create a Work Team Using the Amazon SageMaker Console

You can create a workteam using the Amazon SageMaker console, on the **Labeling workforces** page. You can create a workteam by creating a new Amazon Cognito user group or by importing an existing user group. For more information on creating a user group in the Amazon Cognito console, see Manage a Private Workforce (Amazon Cognito Console) (p. 138).

To create a work team using the Amazon SageMaker console:

1. Open the Amazon SageMaker console at [https://console.aws.amazon.com/sagemaker/](https://console.aws.amazon.com/sagemaker/)
2. Choose **Labeling workforces** from the left menu.
3. Under **Private**, choose **Create private team**.
4. Under **Team details**, enter a **Team name**. The name must be unique in your account in an AWS Region.

5. Under **Add workers**, choose a method to add workers to the team using a user group.
   - If you chose **Create a team by adding workers to a new Amazon Cognito user group**, select the workers to add to the team.
   - If you chose **Create a team by importing existing Amazon Cognito user groups**, choose the user groups that are part of the new team.

6. If you select an **SNS topic**, all workers added to the team are subscribed to the Amazon SNS topic and notified when new work items are available to the team. Select from a list of your existing Ground Truth or Amazon Augmented AI related Amazon SNS topics or select **Create new topic** to open a topic-creation dialog.

Workers in a workteam subscribed to a topic receive notifications when a new job for that team becomes available and when one is about to expire.

Read **Create and manage Amazon SNS topics for your work teams (p. 141)** for more information about using Amazon SNS topic.

**Subscriptions**

After you have created a work team, you can see more information about the team and change or set the Amazon SNS topic to which its members are subscribed by visiting the Amazon Cognito console. Any members of the team who were added to the team prior to the team being subscribed to a topic will need to be subscribed to that topic manually. Read **Create and manage Amazon SNS topics for your work teams** for more information on creating and managing the Amazon SNS topic.

**Add or Remove Workers**

A **work team** is a group of workers within your workforce that you can assign jobs to. A worker can be added to more than one work team. Once a worker has been added to a work team, that worker can be disabled or removed.

**Add Workers to the Workforce**

Adding a worker to the workforce will allow you to add that worker to any work team within that workforce.

**To add workers using the Private workforce summary page:**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
2. Choose **Labeling workforces** to navigate to your Private workforce summary page.
3. Choose **Private**
4. Choose **Invite new workers**
5. Paste or type a list of email addresses, separated by commas, into the email addresses box. You can have up to 50 email addresses in this list.

**Add a Worker to a Work Team**

A worker must be added to the workforce before being added to a work team. To add a worker to a work team, first navigate to the **Private workforce summary** page using the steps above.

**To add a worker to a work team from the workforce summary page**

1. In the **Private teams** section, choose the team that you want to add the workers to
2. Choose the **Workers** tab.
3. Choose **Add workers to team** and choose the boxes next to the workers that you want to add.
4. Click **Add workers to team**.

### Disable and Remove a Worker from the Workforce

Disabling a worker stops the worker from receiving a job. This action does not remove the worker from the workforce, or any work team the worker is associated with. To disable or remove a worker to a work team, first navigate to the Private workforce summary page using the steps above.

**To deactivate a worker using the Private workforce summary page**

1. In the **Workers** section, choose the worker that you would like to disable.
2. Choose **Disable**.

If desired, you can subsequently **Enable** a worker after they have been disabled.

You can remove workers from your private workforce directly in the Amazon SageMaker console if that worker was added in this console. If you added the worker (user) in the Amazon Cognito console, see **Manage a Private Workforce (Amazon Cognito Console)** (p. 138) to learn how to remove the worker in the Amazon Cognito console.

**To remove a worker using the Private workforce summary page**

1. In the **Workers** section, choose the worker that you would like to delete.
2. If the worker has not been disabled, choose **Disable**.
3. Select the worker and choose **Delete**.

### Manage a Private Workforce (Amazon Cognito Console)

A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool. Workers correspond to **Amazon Cognito users** within those groups.

After your workforce has been created, you can add work teams and individual workers through the Amazon Cognito console. You can also delete workers from your private workforce and/or remove them from individual teams in the Amazon Cognito console.

**Important**

You can't delete work teams from the Amazon Cognito console. Deleting a Amazon Cognito user group that is associated with a Amazon SageMaker work team will result in an error. To remove work teams, use the Amazon SageMaker console.

**Create Work Teams (Amazon Cognito Console)**

You can create a new work team to complete a job by adding a Amazon Cognito user group to the user pool associated with your private workforce. To add a Amazon Cognito user group to an existing worker pool, see **Adding groups to a User Pool**.

**To create a work team using an existing Amazon Cognito user group**

2. In the navigation pane, choose **Workforces**.
3. For **Private teams**, choose **Create private team**.
4. Under **Team details**, give the team a name. The name must be unique in your account in an AWS Region.
5. For **Add workers**, choose **Import existing Amazon Cognito user groups**, and choose one or more user groups that are part of the new team.

6. If you choose an **SNS topic**, all workers added to the team are subscribed to the Amazon Simple Notification Service (Amazon SNS) topic and notified when new work items are available to the team. Choose from a list of your existing SNS topics related to Amazon SageMaker Ground Truth or Amazon Augmented AI or choose **Create new topic** to create one.

**Subscriptions**

After you have created a work team, you can see more information about the team and change or set the SNS topic to which its members are subscribed using the Amazon Cognito console. Any members of the team who were added to the team prior to the team being subscribed to a topic need to be subscribed to that topic manually. For information, see Create and manage Amazon SNS topics for your work teams (p. 141).

**Add and Remove Workers (Amazon Cognito Console)**

When using the Amazon Cognito console to add workers to a work team, you must add a user to the user pool associated with the workforce before adding that user to a user group. Users can be added to a user pool in various ways. For more information, see Signing Up and Confirming User Accounts.

**Add a Worker to a Work Team**

After a user has been added to a pool, the user can be associated with user groups inside of that pool. After a user has been added to a user group, that user becomes a worker on any work team created using that user group.

**To add a user to a user group**

1. Open the Amazon Cognito console: https://us-east-2.console.aws.amazon.com/Amazon Cognito/home
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups** and do one of the following:
   - Choose **Groups**, choose the group that you want to add the user to, and choose **Add users**. Choose the users that you want to add by choosing the plus-icon to the right of the users’ name.
   - Choose **Users**, choose the user that you want to add to the user group, and choose **Add to group**. From the drop down menu, choose the group and choose **Add to group**.

**Disable and Remove a Worker From a Work Team**

Disabling a worker stops the worker from receiving a jobs. This action doesn’t remove the worker from the workforce, or any work team the worker is associated with. To remove a user from a work team in Amazon Cognito, you remove the user from the user group associated with that team.

**To deactivate a worker (Amazon Cognito console)**

1. Open the Amazon Cognito console: https://us-east-2.console.aws.amazon.com/Amazon Cognito/home.
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. Choose the user that you want to disable.
Use a Private Workforce

6. Choose **Disable User**

You can enable a disabled user by choosing **Enable User**.

**To remove a user from a user group (Amazon Cognito console)**

1. Open the Amazon Cognito console: https://us-east-2.console.aws.amazon.com/cognito/home.
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. For **User** tab, choose the x-icon to the right of the group that you want to remove the user from.

**Track Worker Performance**

Amazon SageMaker Ground Truth logs worker events to Amazon CloudWatch, such as when a worker starts or submits a task. Use Amazon CloudWatch metrics to measure and track throughput across a team or for individual workers.

**Important**
Worker event tracking is not available for Amazon Augmented AI human review workflows.

**Enable Tracking**

During the set-up process for a new work team, the permissions for Amazon CloudWatch logging of worker events are created. Since this feature was added in August of 2019, work teams created prior to that may not have the correct permissions. If all of your work teams were created before August 2019, create a new work team. It does not need any members and may be deleted after creation, but by creating it, the permissions will be established and will apply to all of your work teams, regardless of when they were created.

**Examine Logs**

After tracking is enabled, the activity of your workers is logged. Open the Amazon CloudWatch console and choose **Logs** in the navigation pane. You should see a log group named **/aws/sagemaker/groundtruth/WorkerActivity**.

Each completed task is represented by a log entry, which contains information about the worker, their team, the job, when the task was accepted, and when it was submitted.

**Example Log entry**

```
{
  "worker_id": "cd449a289e129409",
  "cognito_user_pool_id": "us-east-2_IpicJXXXX",
  "cognito_sub_id": "d6947ae6-0650-447a-ab5d-894db61017fd",
  "task_accepted_time": "Wed Aug 14 16:00:59 UTC 2019",
  "task_submitted_time": "Wed Aug 14 16:01:04 UTC 2019",
  "task_returned_time": "",
  "workteam_arn": "arn:aws:sagemaker:us-east-2:############:workteam/private-crowd/Sample-labeling-team",
  "labeling_job_arn": "arn:aws:sagemaker:us-east-2:############:labeling-job/metrics-demo",
  "work_requester_account_id": "############",
  "job_reference_code": "############",
  "job_type": "Private",
  "event_type": "TasksSubmitted",
  "event_timestamp": "1565798464"
}
```
A useful data point in each event is the `cognito_sub_id`. You can match that to an individual worker.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Under the **Ground Truth** section, choose **Workforces**.
3. Choose **Private**.
4. Choose the name of a team in the **Private teams** section.
5. In the **Team summary** section, choose the user group identified under **Amazon Cognito user group**. That will take you to the group in the Amazon Cognito console.
6. The **Group** page lists the users in the group. Choose any user's link in the **Username** column to see more information about the user, including a unique sub ID.

To get information about all of the team's members, use the **ListUsers** action (examples) in the Amazon Cognito API.

**Use Log Metrics**

If you don't want to write your own scripts to process and visualize the raw log information, Amazon CloudWatch metrics provide insights into worker activity for you.

**To view metrics**

2. In the navigation pane, choose **Metrics**.
3. Choose the **AWS/SageMaker/Workteam** name space, then explore the available metrics (p. 712). For example, selecting the **Workflow, Workteam** metrics lets you calculate the average time per submitted task for a specific labeling job.

For more information, see Using Amazon CloudWatch Metrics.

**Create and manage Amazon SNS topics for your work teams**

Use the procedures in this topic when you need to:

- Create a topic that you want an existing work team to subscribe to.
- Create a topic before you've created a work team.
- Create or modify the work team with an API call, and you need to specify a topic Amazon Resource Name (ARN).

If you create a work team using the console, the console provides an option to create a new topic for the team so that you don't have to perform these steps.

**Create the Amazon SNS topic**

The steps for creating Amazon SNS topics for work team notifications is similar to the steps in Getting Started in the Amazon SNS Developer Guide, with one significant addition—you must add an access policy so that Amazon SageMaker can publish messages to the topic on your behalf.

**To add the policy when you create the topic**

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/.
2. In **Create topic**, enter the name of your topic and then choose **Next steps**.
3. In **Access policy**, choose **Advanced**.
4. In the **JSON editor**, find the **Resource** property, which displays the topic's ARN.
5. Copy the Resource ARN value.
6. Before the final closing brace (}), add the following policy.

```json
,
  "Sid": "AwsSagemaker_SnsAccessPolicy",
  "Effect": "Allow",
  "Principal": {
    "Service": "sagemaker.amazonaws.com"
  },
  "Action": "sns:Publish",
  "Resource": "ARN of the topic you copied in the previous step"
}
```

7. Create the topic.

After you create the topic, it appears in your Topics summary screen. For more information about creating topics, see Creating a Topic in the Amazon SNS Developer Guide.

**Manage worker subscriptions**

If you subscribe a work team to a topic after you've already created the work team, the individual work team members who were added to the team when the work team was created are not automatically subscribed to the topic. For information about subscribing workers' email addresses to the topic, see Subscribing an Endpoint to an Amazon SNS Topic in the Amazon SNS Developer Guide.

The only situation where workers are automatically subscribed to your topic is when you create or import an Amazon Cognito user group at the time that you create a work team and you set up the topic subscription when you create that work team. For more information about creating and managing your workteams with Amazon Cognito, see Create Work Teams (Amazon Cognito Console) (p. 138).

**HTML Elements Reference**

Crowd HTML Elements are web components, a web standard that abstracts HTML markup, CSS, and JavaScript functionality into an HTML tag or set of tags. Amazon SageMaker provides customers with the ability to design their own custom task templates in HTML. For more information about how to implement custom templates in Amazon SageMaker Ground Truth, see Creating Custom Labeling Workflows (p. 84). To learn more about custom templates in Amazon Augmented AI, see Create Custom Templates (p. 114).

**Amazon SageMaker crowd-elements**

Following is a list of enhanced HTML elements that make building a custom template easier and provide a familiar UI for workers. These elements are supported in Ground Truth, Augmented AI, and Mechanical Turk.

**Topics**
- crowd-alert (p. 143)
- crowd-badge (p. 144)
- crowd-button (p. 144)
- crowd-bounding-box (p. 145)
- crowd-card (p. 149)
- crowd-checkbox (p. 149)
- crowd-classifier (p. 151)
crowd-entity-annotation (p. 152)
crowd-fab (p. 154)
crowd-form (p. 155)
crowd-icon-button (p. 156)
crowd-image-classifier (p. 156)
crowd-input (p. 159)
crowd-instance-segmentation (p. 161)
crowd-instructions (p. 163)
crowd-keypoint (p. 164)
crowd-modal (p. 166)
crowd-polygon (p. 167)
crowd-radio-button (p. 172)
crowd-radio-group (p. 173)
crowd-semantic-segmentation (p. 174)
crowd-slider (p. 176)
crowd-tab (p. 178)
crowd-tabs (p. 178)
crowd-text-area (p. 178)
crowd-toast (p. 180)
crowd-toggle-button (p. 181)

crowd-alert

A message that alerts the worker to a current situation.

Attributes

The following attributes are supported by this element.

disable

A Boolean switch that, if present, allows the message to be closed by the worker.

type

A string that specifies the type of message to be displayed. The possible values are "info" (the default), "success", "error", and "warning".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)
crowd-badge

An icon that floats over the top right corner of another element to which it is attached.

Attributes

The following attributes are supported by this element.

for

A string that specifies the ID of the element to which the badge is attached.

icon

A string that specifies the icon to be displayed in the badge. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

This attribute overrides the label attribute.

label

The text to display in the badge. Three characters or less is recommended because text that is too large will overflow the badge area. An icon can be displayed instead of text by setting the icon attribute.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 155)
- Child elements: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-button

A styled button that represents some action.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

form-action

A switch that either submits its parent crowd-form (p. 155) element, if set to "submit", or resets its parent <crowd-form> element, if set to "reset".

href

The URL to an online resource. Use this property if you need a link styled as a button.
icon

A string that specifies the icon to be displayed next to the button's text. The string must be the name of an icon from the open-source iron-icons set, which is pre-loaded. For example, to insert the search iron-icon, use the following:

```html
<iron-icon icon="search"></iron-icon>
```

The icon is positioned to either the left or the right of the text, as specified by the icon-align attribute.

To use a custom icon see icon-url.

icon-align

The left or right position of the icon relative to the button's text. The default is "left".

icon-url

A URL to a custom image for the icon. A custom image can be used in place of a standard icon that is specified by the icon attribute.

loading

A Boolean switch that, if present, displays the button as being in a loading state. This attribute has precedence over the disabled attribute if both attributes are present.

target

When you use the href attribute to make the button act as a hyperlink to a specific URL, the target attribute optionally targets a frame or window where the linked URL should load.

variant

The general style of the button. Use "primary" for primary buttons, "normal" for secondary buttons, "link" for tertiary buttons, or "icon" to display only the icon without text.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- **Use Amazon SageMaker Ground Truth for Labeling** (p. 49)
- **HTML Elements Reference** (p. 142)

crowd-bounding-box

A widget for drawing rectangles on an image and assigning a label to the portion of the image that is enclosed in each rectangle.
Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

An array of JSON objects, each of which sets a bounding box when the component is loaded. Each JSON object in the array contains the following properties. Bounding boxes set via the initial-value property can be adjusted and whether or not a worker answer was adjusted is tracked via an initialValueModified boolean in the worker answer output.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the labels attribute of the <crowd-bounding-box> element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

You can extract the bounding box initial value from a manifest file of a previous job in a custom template using the Liquid templating language:

```liquid
initial-value="[%
  {% for box in task.input.manifestLine.label-attribute-name-from-prior-job.annotations %}
    {% capture class_id %}{{ box.class_id }}{% endcapture %}
    {% assign label = task.input.manifestLine.label-attribute-name-from-prior-job-metadata.class-map[class_id] %}
    {%
      label: {{label | to_json}},
      left: {{box.left}},
      top: {{box.top}},
      width: {{box.width}},
      height: {{box.height}},
    },
  {% endfor %}
%]"
```

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a rectangle. **Limit: 10 labels.**

name

The name of this widget. It's used as a key for the widget's input in the form output.

src

The URL of the image on which to draw bounding boxes.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
• **Child elements:** full-instructions (p. 147), short-instructions (p. 147)

**Regions**

The following regions are required by this element.

**full-instructions**

General instructions about how to draw bounding boxes.

**short-instructions**

Important task-specific instructions that are displayed in a prominent place.

**Output**

The following output is supported by this element.

**boundingBoxes**

An array of JSON objects, each of which specifies a bounding box that has been created by the worker. Each JSON object in the array contains the following properties.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the `labels` attribute of the `<crowd-bounding-box>` element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

**inputImageProperties**

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

**Example : Sample Element Outputs**

The following are samples of outputs from common use scenarios for this element.

**Single Label, Single Box / Multiple Label, Single Box**

```json
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
          "height": 401,
          "label": "Dog",
          "left": 243,
          "top": 117,
          "width": 187
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 1200
      }
    }
  ]
```
Single Label, Multiple Box

```
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
          "height": 401,
          "label": "Dog",
          "left": 243,
          "top": 117,
          "width": 187
        },
        {
          "height": 283,
          "label": "Dog",
          "left": 684,
          "top": 120,
          "width": 116
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 800
      }
    }
  }
]
```

Multiple Label, Multiple Box

```
[
  {
    "annotatedResult": {
      "boundingBoxes": [
        {
          "height": 395,
          "label": "Dog",
          "left": 241,
          "top": 125,
          "width": 158
        },
        {
          "height": 298,
          "label": "Cat",
          "left": 699,
          "top": 116,
          "width": 101
        }
      ],
      "inputImageProperties": {
        "height": 533,
        "width": 800
      }
    }
  }
]
```
You could have many labels available, but only the ones that are used appear in the output.

See Also
For more information, see the following.
- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-card
A box with an elevated appearance for displaying information.

Attributes
The following attributes are supported by this element.

heading
The text displayed at the top of the box.

image
A URL to an image to be displayed within the box.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also
For more information, see the following.
- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-checkbox
A UI component that can be checked or unchecked allowing a user to select multiple options from a set.

Attributes
The following attributes are supported by this element.

checked
A Boolean switch that, if present, displays the check box as checked.

disabled
A Boolean switch that, if present, displays the check box as disabled and prevents it from being checked.
name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A string used as the name for the check box state in the output. Defaults to "on" if not specified.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 155)
- Child elements: none

Output

Provides a JSON object. The name string is the object name and the value string is the property name for a Boolean value based on the check box state; true if checked, false if not checked.

Example: Sample Element Outputs

Using the same name value for multiple boxes.

```html
<!-- INPUT -->
<div><crowd-checkbox name="myformbit" value="Red"> Red </div>
<div><crowd-checkbox name="myformbit" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="myformbit" value="Green"> Green </div>

//Output with "Red" checked
[
  {
    "myformbit": {
      "Green": false,
      "Red": true,
      "Yellow": false
    }
  }
]
```

Note that all three color values are properties of a single object.

Using different name values for each box.

```html
<!-- INPUT -->
<div><crowd-checkbox name="Stop" value="Red"> Red </div>
<div><crowd-checkbox name="Slow" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="Go" value="Green"> Green </div>

//Output with "Red" checked
[
]```
{  
  "Go": {  
    "Green": false  
  },  
  "Slow": {  
    "Yellow": false  
  },  
  "Stop": {  
    "Red": true  
  }  
}

See Also
For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-classifier
A widget for classifying non-image content, such as audio, video, or text.

Attributes
The following attributes are supported by this element.

categories
A JSON formatted array of strings, each of which is a category that a worker can assign to the to the text. You should include "other" as a category, otherwise the worker my not be able to provide an answer.

header
The text to display above the image. This is typically a question or simple instruction for the worker.

name
The name of this widget. It is used as a key for the widget's input in the form output.

Element Hierarchy
This element has the following parent and child elements.

- Parent elements: crowd-form (p. 155)
- Child elements: classification-target (p. 151), full-instructions (p. 152), short-instructions (p. 152)

Regions
The following regions are supported by this element.

classification-target
The content to be classified by the worker. This can be plain text or HTML. Examples of how the HTML can be used include but are not limited to embedding a video or audio player, embedding a PDF, or performing a comparison of two or more images.
full-instructions
General instructions about how to do text classification.

short-instructions
Important task-specific instructions that are displayed in a prominent place.

Output
The output of this element is an object using the specified name value as a property name, and a string from the categories as the property's value.

Example: Sample Element Outputs
The following is a sample of output from this element.

```
[
  {
    "<name>": {
      "label": "<value>"
    }
  }
]
```

See Also
For more information, see the following.
- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-entity-annotation
A widget for labeling words, phrases, or character strings within a longer text.

Important: Self-contained Widget
Do not use `<crowd-entity-annotation>` element with the `<crowd-form>` element. It contains its own form submission logic and Submit button.

Attributes
The following attributes are supported by this element.

header
The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value
A JSON formatted array of objects, each of which defines an annotation to apply to the text at initialization. Objects contain a label value that matches one in the labels attribute, an integer startOffset value for labeled span's starting unicode offset, and an integer endOffset value for the ending unicode offset.

Example
```
[
```

labels

A JSON formatted array of objects, each of which contains:

- **label** (required): The name used to identify entities.
- **fullDisplayName** (optional): Used for the label list in the task widget. Defaults to the label value if not specified.
- **shortDisplayName** (optional): An abbreviation of 3-4 letters to display above selected entities. Defaults to the label value if not specified. **shortDisplayName is highly recommended**

Values displayed above the selections can overlap and create difficulty managing labeled entities in the workspace. Providing a 3-4 character shortDisplayName for each label is highly recommended to prevent overlap and keep the workspace manageable for your workers.

Example

```json
[
  {
    label: 'person',
    shortDisplayName: 'per',
    fullDisplayName: 'person'
  }
]
```

name

Serves as the widget's name in the DOM. It is also used as the label attribute name in form output and the output manifest.

text

The text to be annotated. The templating system escapes quotes and HTML strings by default. If your code is already escaped or partially escaped, see Variable filters (p. 89) for more ways to control escaping.

Element Hierarchy

This element has the following parent and child elements.

- **Child elements**: full-instructions (p. 153), short-instructions (p. 154)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to work with the widget.
Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

entities

A JSON object that specifies the start, end, and label of an annotation. This object contains the following properties.

- **label** – The assigned label.
- **startOffset** – The Unicode offset of the beginning of the selected text.
- **endOffset** – The Unicode offset of the first character after the selection.

Example: Sample Element Outputs

The following is a sample of the output from this element.

```json
{
  "myAnnotatedResult": {
    "entities": [
      {
        "endOffset": 54,
        "label": "person",
        "startOffset": 47
      },
      {
        "endOffset": 97,
        "label": "event",
        "startOffset": 93
      },
      {
        "endOffset": 219,
        "label": "date",
        "startOffset": 212
      },
      {
        "endOffset": 271,
        "label": "location",
        "startOffset": 260
      }
    ]
  }
}
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-fab

A floating button with an image in its center.
Attributes

The following attributes are supported by this element.

**disabled**

A Boolean switch that, if present, displays the floating button as disabled and prevents clicks.

**icon**

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

**label**

A string consisting of a single character that can be used instead of an icon. Emojis or multiple characters may result in the button displaying an ellipsis instead.

**title**

A string that will display as a tool tip when the mouse hovers over the button.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-form**

The form wrapper for all custom tasks. Sets and implements important actions for the proper submission of your form data.

If a crowd-button (p. 144) of type "submit" is not included inside the <crowd-form> element, it will automatically be appended within the <crowd-form> element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: none
- **Child elements**: Any of the UI Template (p. 142) elements

Element Events

The crowd-form element extends the standard HTML form element and inherits its events, such as onclick and onsubmit.
See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-icon-button

A button with an image placed in the center. When the user touches the button, a ripple effect emanates from the center of the button.

Attributes

The following attributes are supported by this element.

- **disabled**
  A Boolean switch that, if present, displays the button as disabled and prevents clicks.

- **icon**
  A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-image-classifier

A widget for classifying an image, which can be a JPG, PNG, or GIF, with no size limit.

Attributes

The following attributes are required by this element.

- **categories**
  A JSON formatted array of strings, each of which is a category that a worker can assign to the image. You should include "other" as a category, so that the worker can provide an answer. You can specify up to 10 categories.

- **header**
  The text to display above the image. This is typically a question or simple instruction for the worker.
name

The name of this widget. It is used as a key for the widget's input in the form output.

overlay

Information to be overlaid on the source image. This is for verification workflows of bounding-box and semantic-segmentation tasks.

It is a JSON object containing an object with the name of the task-type in camelCase as the key. That key's value is an object that contains the labels and other necessary information from the previous task.

An example of a crowd-image-classifier element with attributes for verifying a semantic segmentation task follows:

```xml
<crowd-image-classifier
    name='crowd-image-classifier'
    categories='["good", "bad"]'
    src='URL of image to be classified'
    header='Please classify'
    overlays='{
        "semanticSegmentation": {
            "labels": ["Cat", "Dog", "Bird", "Cow"],
            "labelMappings": {
                "Bird": {
                    "color": "#ff7f0e"
                },
                "Cat": {
                    "color": "#2ca02c"
                },
                "Cow": {
                    "color": "#d62728"
                },
                "Dog": {
                    "color": "#2acf59"
                }
            },
            "src": "URL of overlay image",
        }
    }
'>

A bounding-box verification task would use the overlay value like follows:

```xml
<crowd-image-classifier
    name="boundingBoxClassification"
    header="Rate the quality of the annotations based on the background section in the instructions on the left hand side."
    src="https://i.imgur.com/CIPKVJo.jpg"
    categories=['good', 'bad', 'okay']
    overlays='{
        "boundingBox": {
            "labels": ["bird", "cat"],
            value: [
                { height: 284,
                  label: "bird",
                  left: 230,
                  top: 974,
                  width: 223
                },
                { height: 69,
                  label: "cat",
                  left: 129,
                  top: 968,
                  width: 123
                }
            ]
        }
    }
'>
The URL of the image to be classified.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: full-instructions (p. 158), short-instructions (p. 158), worker-comment (p. 158)

**Regions**

The following regions are used by this element.

**full-instructions**

General instructions for the worker on how to classify an image.

**short-instructions**

Important task-specific instructions that are displayed in a prominent place.

**worker-comment**

Use this in verification workflows when you need workers to explain why they made the choice they did. Use the text between the opening and closing tags to provide instructions for workers on what information should be included in the comment.

It uses the following attributes:

**header**

A phrase with a call to action for leaving a comment. Used as the title text for a modal window where the comment is added.

Optional. Defaults to "Add a comment."

**link-text**

This text appears below the categories in the widget. When clicked, it opens a modal window where the worker may add a comment.

Optional. Defaults to "Add a comment."

**placeholder**

An example text in the comment text area that is overwritten when worker begins to type. This does not appear in output if the worker leaves the field blank.
Optional. Defaults to blank.

Output

The output of this element is a string that specifies one of the values defined in the categories attribute of the <crowd-image-classifier> element.

Example: Sample Element Outputs

The following is a sample of output from this element.

```
[
  {
    "<name>": {
      "label": "<value>",
      "workerComment": "Comment - if no comment is provided, this field will not be present"
    }
  }
]
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-input

A box that accepts input data.

Cannot be self-closing

Unlike the input element in the HTML standard, this element cannot be self-closed by putting a slash before the ending bracket, e.g. `<crowd-input ... />`. It must be followed with a `</crowd-input>` to close the element.

Attributes

The following attributes are supported by this element.

allowed-pattern

A regular expression that is used with the auto-validate attribute to ignore non-matching characters as the worker types.

auto-focus

When the value is set to true, the browser places focus inside the input area after loading. This way, the worker can start typing without having to select it first.

auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the error-message and allowed-pattern attributes.

disabled

A Boolean switch that, if present, displays the input area as disabled.
error-message

The text to be displayed below the input field, on the left side, if validation fails.

label

A string that is displayed inside a text field.
This text shrinks and rises up above a text field when the worker starts typing in the field or when the value attribute is set.

max-length

A maximum number of characters the input will accept. Input beyond this limit is ignored.

min-length

A minimum length for the input in the field

name

Sets the name of the input to be used in the DOM and the output of the form.

placeholder

A string value that is used as placeholder text, displayed until the worker starts entering data into the input, It is not used as a default value.

required

A Boolean switch that, if present, requires the worker to provide input.

type

Takes a string to set the HTML5 input-type behavior for the input. Examples include file and date.

value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 155)
- Child elements: none

Output

Provides a name string as the property name, and the text that was entered in the field as its value.

Example: Sample JSON Output

The values for multiple elements are output in the same object, with their name attribute value as their property name. Elements with no input do not appear in the output. For example, let's use three inputs:

```xml
<crowd-input name="tag1" label="Word/phrase 1"></crowd-input>
<crowd-input name="tag2" label="Word/phrase 2"></crowd-input>
<crowd-input name="tag3" label="Word/phrase 3"></crowd-input>
```
This is the output if only two have input:

```json
[
  {
    "tag1": "blue",
    "tag2": "red"
  }
]
```

This means any code built to parse these results should be able to handle the presence or absence of each input in the answers.

**See Also**

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-instance-segmentation**

A widget for identifying individual instances of specific objects within an image and creating a colored overlay for each labeled instance.

**Attributes**

The following attributes are supported by this element.

- **header**
  The text to display above the image. This is typically a question or simple instruction for the worker.

- **labels**
  A JSON formatted array of strings, each of which is a label that a worker can assign to an instance of an object in the image. Workers can generate different overlay colors for each relevant instance by selecting "add instance" under the label in the tool.

- **name**
  The name of this widget. It is used as a key for the labeling data in the form output.

- **src**
  The URL of the image that is to be labeled.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: full-instructions (p. 162), short-instructions (p. 162)

**Regions**

The following regions are supported by this element.
full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

instances

A JSON Array containing objects with the instance labels and colors.

• **color** – The hexadecimal value of the label's RGB color in the labeledImage PNG.
• **label** – The label given to overlay(s) using that color. This value may repeat, because the different instances of the label are identified by their unique color.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

• **height** – The height, in pixels, of the image.
• **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```json
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 533,
        "width": 800
      },
      "instances": [
        {
          "color": "#1f77b4",
          "label": "<Label 1>"
        },
        {
          "color": "#2ca02c",
          "label": "<Label 1>"
        },
        {
          "color": "#ff7f0e",
          "label": "<Label 3>"
        }
      ],
      "labeledImage": {
        "pngImageData": "<Base-64 Encoded Data>"
      }
    }
  }
]```
See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-instructions**

An element that displays instructions on three tabbed pages, **Summary**, **Detailed Instructions**, and **Examples**, when the worker clicks on a link or button.

**Attributes**

The following attributes are supported by this element.

- **link-text**

  The text to display for opening the instructions. The default is **Click for instructions**.

- **link-type**

  A string that specifies the type of trigger for the instructions. The possible values are "link" (default) and "button".

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

**Regions**

The following regions are supported by this element.

- **detailed-instructions**

  Content that provides specific instructions for a task. This appears on the page of the "Detailed Instructions" tab.

- **negative-example**

  Content that provides examples of inadequate task completion. This appears on the page of the "Examples" tab. More than one example may be provided within this element.

- **positive-example**

  Content that provides examples of proper task completion. This appears on the page of the "Examples" tab.
short-summary

A brief statement that summarizes the task to be completed. This appears on the page of the "Summary" tab. More than one example may be provided within this element.

See Also

For more information, see the following.
- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-keypoint

Generates a tool to select and annotate key points on an image.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

An array, in JSON format, of keypoints to be applied to the image on start. For example:

```json
initial-value="[
  {
    'label': 'Left Eye',
    'x': 1022,
    'y': 429
  },
  {
    'label': 'Beak',
    'x': 941,
    'y': 403
  }
]
```

Note

Please note that label values used in this attribute must have a matching value in the labels attribute or the point will not be rendered.

labels

An array, in JSON format, of strings to be used as keypoint annotation labels.

name

A string used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

src

The source URI of the image to be annotated.
Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: full-instructions (p. 165), short-instructions (p. 165)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to annotate the image.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

keypoints

An array of JSON objects containing the coordinates and label of a keypoint. Each object contains the following properties.

- **label** – The assigned label for the keypoint.
- **x** – The X coordinate, in pixels, of the keypoint on the image.
- **y** – The Y coordinate, in pixels, of the keypoint on the image.

**Note**

X and Y coordinates are based on 0,0 being the top left corner of the image.

Example: Sample Element Outputs

The following is a sample output from using this element.

```json
[
  {
    "crowdKeypoint": {
      "inputImageProperties": {
        "height": 1314,
        "width": 962
      },
      "keypoints": [
```
You may have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

Crowd-modal

A small window that pops up on the display when it is opened.

Attributes

The following attributes are supported by this element.

- link-text

The text to display for opening the modal. The default is "Click to open modal".

- link-type

A string that specifies the type of trigger for the modal. The possible values are "link" (default) and "button".
Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-polygon**

A widget for drawing polygons on an image and assigning a label to the portion of the image that is enclosed in each polygon.

**Attributes**

The following attributes are supported by this element.

- **header**
  The text to display above the image. This is typically a question or simple instruction for the worker.

- **labels**
  A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a polygon.

- **name**
  The name of this widget. It's used as a key for the widget's input in the form output.

- **src**
  The URL of the image on which to draw polygons.

- **initial-value**
  An array of JSON objects, each of which defines a polygon to be drawn when the component is loaded. Each JSON object in the array contains the following properties.

  - **label** – The text assigned to the polygon as part of the labeling task. This text must match one of the labels defined in the `labels` attribute of the `<crowd-polygon>` element.
  
  - **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon.

**Example**

An `initial-value` attribute might look something like this.

```json
initial-value =
  '{

```
"label": "dog",
"vertices":
[
  {
    "x": 570,
    "y": 239
  },
  ...
  {
    "x": 759,
    "y": 281
  }
]}

Because this will be within an HTML element, the JSON array must be enclosed in single or double quotes. The example above uses single quotes to encapsulate the JSON and double quotes within the JSON itself. If you must mix single and double quotes inside your JSON, replace them with their HTML entity codes (&quot; for double quote, &apos; for single) to safely escape them.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: full-instructions (p. 168), short-instructions (p. 168)

**Regions**

The following regions are required.

- **full-instructions**
  General instructions about how to draw polygons.

- **short-instructions**
  Important task-specific instructions that are displayed in a prominent place.

**Output**

The following output is supported by this element.

- **polygons**
  An array of JSON objects, each of which describes a polygon that has been created by the worker. Each JSON object in the array contains the following properties.

  - **label** – The text assigned to the polygon as part of the labeling task.
  - **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon. The top left corner of the image is 0,0.

- **inputImageProperties**
  A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

  - **height** – The height, in pixels, of the image.
• **width** – The width, in pixels, of the image.

**Example: Sample Element Outputs**

The following are samples of outputs from common use scenarios for this element.

**Single Label, Single Polygon**

```json
{
  "annotatedResult": {
    "inputImageProperties": {
      "height": 853,
      "width": 1280
    },
    "polygons": [
      {
        "label": "dog",
        "vertices": [
          {
            "x": 570,
            "y": 239
          },
          {
            "x": 603,
            "y": 513
          },
          {
            "x": 823,
            "y": 645
          },
          {
            "x": 901,
            "y": 417
          },
          {
            "x": 759,
            "y": 281
          }
        ]
      }
    ]
  }
}
```

**Single Label, Multiple Polygons**

```json
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons": [
        {
          "label": "dog",
          "vertices": [
            {
              "x": 570,
              "y": 239
            }
          ]
        }
      ]
    }
  }
]```
Multiple Labels, Multiple Polygons

[

]
{"annotatedResult": {
  "inputImageProperties": {
    "height": 853,
    "width": 1280
  },
  "polygons": [
    {
      "label": "dog",
      "vertices": [
        {
          "x": 570,
          "y": 239
        },
        {
          "x": 603,
          "y": 513
        },
        {
          "x": 823,
          "y": 645
        },
        {
          "x": 901,
          "y": 417
        },
        {
          "x": 759,
          "y": 281
        }
      ]
    },
    {
      "label": "cat",
      "vertices": [
        {
          "x": 870,
          "y": 278
        },
        {
          "x": 908,
          "y": 446
        },
        {
          "x": 1009,
          "y": 602
        },
        {
          "x": 1116,
          "y": 519
        },
        {
          "x": 1174,
          "y": 498
        },
        {
          "x": 1227,
          "y": 479
        },
        {
          "x": 1179,
          "y": 405
        },
        {
          "x": 1179,
          "y": 337
        }
      ]
    }
  ]
}
You could have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-radio-button**

A button that can be either checked or unchecked. When radio buttons are inside a radio group, exactly one radio button in the group can be checked at any time. For an example of a custom UI template that uses the crowd-radio-button element, see this entity recognition labeling job custom template.

Attributes

The following attributes are supported by this element.

- **checked**
  A Boolean switch that, if present, displays the radio button as checked.

- **disabled**
  A Boolean switch that, if present, displays the button as disabled and prevents it from being checked.

- **name**
  A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

  **Note**
  If you use the buttons outside of a crowd-radio-group (p. 173) element, but with the same name string and different value strings, the name object in the output will contain a Boolean value for each value string. To ensure that only one button in a group is selected, make them children of a crowd-radio-group (p. 173) element and use different name values.

- **value**
  A property name for the element's boolean value. If not specified, it uses "on" as the default, e.g. 
  
  ```
  { "<name>": { "<value>": <true or false> } }
  ```

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-radio-group (p. 173)
- **Child elements**: none
Output

Outputs an object with the following pattern: `{ "<name>": { "<value>": <true or false> } }`. If you use the buttons outside of a crowd-radio-group (p. 173) element, but with the same name string and different value strings, the name object will contain a Boolean value for each value string. To ensure that only one in a group of buttons is selected, make them children of a crowd-radio-group (p. 173) element and use different name values.

Example Sample output of this element

```
[

  { "btn1": { "yes": true },
    "btn2": { "no": false }
  }
]
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-radio-group

A group of radio buttons. Only one radio button within the group can be selected. Choosing one radio button clears any previously chosen radio button within the same group. For an example of a custom UI template that uses the crowd-radio-group element, see this entity recognition labeling job custom template.

Attributes

No special attributes are supported by this element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: crowd-radio-button (p. 172)

Output

Outputs an array of objects representing the crowd-radio-button (p. 172) elements within it.

Example Sample of Element Output

```
[

  { "btn1": { "yes": true
```
crowd-semantic-segmentation

A widget for segmenting an image and assigning a label to each image segment.

Attributes

The following attributes are supported by this element.

**header**

The text to display above the image. This is typically a question or simple instruction for the worker.

**initial-value**

A JSON object containing the color mappings of a prior semantic segmentation job and a link to the overlay image output by the prior job. Include this when you want a human worker to verify the results of a prior labeling job and adjust it if necessary.

The attribute would appear as follows:

```diablo
initial-value='{
  "labelMappings": {
    "Bird": {
      "color": "#ff7f0e"
    },
    "Cat": {
      "color": "#2ca02c"
    },
    "Cow": {
      "color": "#d62728"
    },
    "Dog": {
      "color": "#1f77b4"
    }
  },
  "src": {{ "S3 file URL for image" | grant_read_access }}
}
```

While label mappings are included in individual worker output records, the overall result is represented as the **internal-color-map** in the consolidated results.

You can convert the **internal-color-map** to **label-mappings** in a custom template using the Liquid templating language:

```diablo
initial-value="{
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)
labels

A JSON formatted array of strings, each of which is a label that a worker can assign to a segment of the image.

name

The name of this widget. It is used as a key for the widget's input in the form output.

ts

The URL of the image that is to be segmented.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 155)
- Child elements: full-instructions (p. 175), short-instructions (p. 175)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

labelMappings

A JSON Object containing objects with named with the segmentation labels.

- color – The hexadecimal value of the label's RGB color in the labeledImage PNG.
initialValueModified

A boolean representing whether the initial values have been modified. This is only included when the output is from an adjustment task.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- height – The height, in pixels, of the image.
- width – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```json
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 533,
        "width": 800
      },
      "labelMappings": {
        "<label 2>": {
          "color": "#ff7f0e"
        },
        "<label 3>": {
          "color": "#2ca02c"
        },
        "<label 1>": {
          "color": "#1f77b4"
        }
      },
      "labeledImage": {
        "pngImageData": "<Base-64 Encoded Data>"
      }
    }
  }
]
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

crowd-slider

A bar with a sliding knob that allows a worker to select a value from a range of values by moving the knob. The slider makes it a great choice for settings that reflect intensity levels, such as volume, brightness, or color saturation.

Attributes

The following attributes are supported by this element.
**disabled**

A Boolean switch that, if present, displays the slider as disabled.

**editable**

A Boolean switch that, if present, displays an up/down button that can be chosen to select the value.

Selecting the value via the up/down button is an alternative to selecting the value by moving the knob on the slider. The knob on the slider will move synchronously with the up/down button choices.

**max**

A number that specifies the maximum value on the slider.

**min**

A number that specifies the minimum value on the slider.

**name**

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

**pin**

A Boolean switch that, if present, displays the current value above the knob as the knob is moved.

**required**

A Boolean switch that, if present, requires the worker to provide input.

**secondary-progress**

When used with a `crowd-slider-secondary-color` CSS attribute, the progress bar is colored to the point represented by the `secondary-progress`. For example, if this was representing the progress on a streaming video, the `value` would represent where the viewer was in the video timeline. The `secondary-progress` value would represent the point on the timeline to which the video had buffered.

**step**

A number that specifies the difference between selectable values on the slider.

**value**

A preset that becomes the default if the worker does not provide input.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: `crowd-form` (p. 155)
- **Child elements**: none

**See Also**

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)
**crowd-tab**

A component styled to look like a tab with information below.

**Attributes**

The following attributes are supported by this element.

**header**

The text appearing on the tab. This is usually some short descriptive name indicative of the information contained below the tab.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-tabs (p. 178)
- **Child elements**: none

**See Also**

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-tabs**

A container for tabbed information.

**Attributes**

This element has no attributes.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: crowd-tab (p. 178)

**See Also**

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-text-area**

A field for text input.
Attributes

The following attributes are supported by this element.

auto-focus

A Boolean switch that, if present, puts the cursor in this element on-load so that users can immediately begin typing without having to click inside the element.

auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the error-message and allowed-pattern attributes.

char-counter

A Boolean switch that, if present, puts a small text field beneath the lower-right corner of the element, displaying the number of characters inside the element.

disabled

A Boolean switch that, if present, displays the input area as disabled.

error-message

The text to be displayed below the input field, on the left side, if validation fails.

label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the value attribute is set.

max-length

An integer that specifies the maximum number of characters allowed by the element. Characters typed or pasted beyond the maximum are ignored.

max-rows

An integer that specifies the maximum number of rows of text that are allowed within a crowd-text-area. Normally the element expands to accommodate new rows. If this is set, after the number of rows exceeds it, content scrolls upward out of view and a scrollbar control appears.

name

A string used to represent the element's data in the output.

placeholder

A string presented to the user as placeholder text. It disappears after the user puts something in the input area.

rows

An integer that specifies the height of the element in rows of text.

value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.
Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

Output

This element outputs the `name` as a property name and the element's text contents as the value. Carriage returns in the text are represented as `\n`.

**Example Sample output for this element**

```json
[
  {
    "textInput1": "This is the text; the text that\nmakes the crowd go wild."
  }
]
```

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)

**crowd-toast**

A subtle notification that temporarily appears on the display. Only one crowd-toast is visible.

**Attributes**

The following attributes are supported by this element.

- **duration**
  
  A number that specifies the duration, in seconds, that the notification appears on the screen.

- **text**
  
  The text to display in the notification.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

See Also

For more information, see the following.

- Use Amazon SageMaker Ground Truth for Labeling (p. 49)
- HTML Elements Reference (p. 142)
crowd-toggle-button

A button that acts as an ON/OFF switch, toggling a state.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the button switched to the ON position.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents toggling.

invalid

When in an off position, a button using this attribute, will display in an alert color. The standard is red, but may be changed in CSS. When toggled on, the button will display in the same color as other buttons in the on position.

name

A string that is used to identify the answer submitted by the worker. This value matches a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A value used in the output as the property name for the element's Boolean state. Defaults to "on" if not provided.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 155)
- **Child elements**: none

Output

This element outputs the name as the name of an object, containing the value as a property name and the element's state as Boolean value for the property. If no value for the element is specified, the property name defaults to "on."

Example Sample output for this element

```
[
  {
    "theToggler": {
      "on": true
    }
  }
]
```
Augmented AI crowd-elements

The following HTML elements are only available for Amazon Augmented AI human workflow tasks.

Topics
- crowd-textract-document-analysis (p. 182)
- crowd-rekognition-detect-moderation-labels (p. 186)

crowd-textract-document-analysis

A widget to enable human review of an Amazon Textract document analysis result.

Attributes
The following attributes are supported by this element.

header
This is the text that is displayed as the header.

src
This is a link to the image to be analyzed by the worker.

initialValue
This sets initial values for attributes found in the worker UI.

The following is an example of an initialValue input:

```json
[
  {
    "BlockType": "KEY_VALUE_SET",
    "Confidence": 38.43309020996094,
    "Geometry": {
      "BoundingBox": {
        "Width": 0.32613086700439453,
        "Height": 0.0942094624042511,
        "Left": 0.4833833575248718,
        "Top": 0.5227988958358765
      },
      "Polygon": [
        {"X": 0.123, "Y": 0.345}, ...
      ]
    },
    "Id": "8c97b240-0969-4678-834a-646c95d9cf4",
    "Relationships": [
      {"Type": "CHILD",
      "Ids": [
```
blockTypes
This determines the kind of analysis the workers can do. Only KEY.VALUE_SET is currently supported.

keys
This specifies new keys and the associated text value the worker can add. The input values for keys can include the following elements:

- importantFormKey accepts strings, and is used to specify a single key.
- importantFormKeyAliases can be used to specify aliases that are acceptable alternatives to the keys supplied. Use this element to identify alternative spellings or presentations of your keys. This parameter accepts a list of one or more strings.

The following is an example of an input for keys.

[ 
  { 
    importantFormKey: 'Address', 
    importantFormKeyAliases: [ 
      'address', 
      'Addr.', 
      'Add.' 
    ] 
  }, 
  { 
    importantFormKey: 'Last name', 
    importantFormKeyAliases: ['Surname'] 
  } 
]

no-key-edit
This prevents the workers from editing the keys of annotations passed through initialValue. If you want to prevent workers from editing the keys that have been detected on your documents, you should include this attribute.

no-geometry-edit
This prevents workers from editing the polygons of annotations passed through initialValue. For example, this would prevent the worker from editing the bounding box around a given key. In most scenarios, you should include this attribute.
Element Hierarchy

This element has the following parent and child elements.

- Parent elements – crowd-form
- Child elements – full-instructions (p. 184), short-instructions (p. 184)

AWS Regions

The following AWS Regions are supported by this element. You can use custom HTML and CSS code within these Regions to format your instructions to workers. For example, use the short-instructions section to provide good and bad examples of how to complete a task.

full-instructions

General instructions about how to work with the widget.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Example of a Worker Template Using the crowd Element

An example of a worker template using this crowd element would look like the following.

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_arn %}http://s3.amazonaws.com/{{ task.input.aiServiceRequest.document.s3Object.bucket }}/{{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-textract-analyze-document
    src="{{ s3_arn | grant_read_access }}"
    initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
    header="Review the key-value pairs listed on the right and correct them if they don't match the following document."
    no-key-edit
    no-geometry-edit
    keys="{{ task.input.humanLoopContext.importantFormKeys }}"
    block-types="["KEY_VALUE_SET"]"
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
        .instructionsImage {
          display: inline-block;
          max-width: 100%;
        }
      </style>
      <p class='instructions'>Click on a key-value block to highlight the corresponding key-value pair in the document.

      If it is a valid key-value pair, review the content for the value. If the content is incorrect, correct it.
      The text of the value is incorrect, correct it.
      <img class='instructionsImage' src="https://assets.crowd.aws/images/a2i-console/correct-value-text.png" />
    </short-instructions>
  </crowd-textract-analyze-document>
</crowd-form>
```
A wrong value is identified, correct it.
If it is not a valid key-value relationship, choose No.
If you can’t find the key in the document, choose Key not found.
If the content of a field is empty, choose Value is blank.
Key and value are often displayed next or below to each other.
Key and value displayed in one line.
Key and value displayed in two lines.
If the content of the value has multiple lines, enter all the text without line break.
Include all value text even if it extends beyond the highlight box.

```
{ "AWS/Textract/AnalyzeDocument/Forms/V1": { 
  "blocks": [ 
    { "blockType": "KEY_VALUE_SET", 
      "id": "8c97b240-0969-4678-834a-646c95da9cf4", 
      "relationships": [ 
        { "type": "CHILD", 
          "ids": ["7ee7b7da-ee1b-428d-a567-55a3e3affa56", "4d6da730-ba43-467c-a9a5-c6137ba0c472"] } 
      ] 
    }, 
    { "type": "VALUE", 
      "ids": ["6ee7b7da-ee1b-428d-a567-55a3e3affa54"] } 
  ], 
  "entityTypes": ["KEY"], 
  "text": "Foo bar baz" 
} 
}
```

Output

The following is a sample of the output from this element. You can find a detailed explanation of this output in the Amazon Textract AnalyzeDocument API documentation.
crowd-rekognition-detect-moderation-labels

A widget to enable human review of an Amazon Rekognition image moderation result.

Attributes

The following attributes are supported by this element.

header
This is the text that is displayed as the header.

src
This is a link to the image to be analyzed by the worker.

categories
This supports categories as an array of strings or an array of objects where each object has a name field.

If the categories come in as objects, the following applies:

• The displayed categories are the value of the name field.
• The returned answer contains the full objects of any selected categories.

If the categories come in as strings, the following applies:

• The returned answer is an array of all the strings that were selected.

exclusion-category
By setting this attribute you create a button underneath the categories in the UI.

• When a user chooses the button, all categories are deselected and disabled.
• Choosing the button again re-enables the categories so that users can choose them.
• If you submit after choosing the button, it returns an empty array.

Element Hierarchy

This element has the following parent and child elements.

• Parent elements – crowd-form
• Child elements – full-instructions (p. 186), short-instructions (p. 187)

AWS Regions

The following AWS Regions are supported by this element. You can use custom HTML and CSS code within these Regions to format your instructions to workers. For example, use the short-instructions section to provide good and bad examples of how to complete a task.

full-instructions
General instructions about how to work with the widget.
short-instructions

Important task-specific instructions that are displayed in a prominent place.

Example Worker Template with the crowd Element

An example of a worker template using the crowd element would look like the following.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_arn %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/  
{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
      categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
      {
        name: "{{ label.name }}",
        parentName: "{{ label.parentName }}",
      },
      {% endfor %}
      ]'
      src="{{ s3_arn | grant_read_access }}"
      header="Review the image and choose all applicable categories."
    >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
      </style>
      <p class='instructions'>Review the image and choose all applicable categories. If no categories apply, choose None.

    <b>Nudity</b>
    Visuals depicting nude male or female person or persons

    <b>Graphic Male Nudity</b>
    Visuals depicting full frontal male nudity, often close ups

    <b>Graphic Female Nudity</b>
    Visuals depicting full frontal female nudity, often close ups

    <b>Sexual Activity</b>
    Visuals depicting various types of explicit sexual activities and pornography

    <b>Illustrated Nudity or Sexual Activity</b>
    Visuals depicting animated or drawn sexual activity, nudity or pornography

    <b>Adult Toys</b>
    Visuals depicting adult toys, often in a marketing context

    <b>Female Swimwear or Underwear</b>
    Visuals depicting female person wearing only swimwear or underwear

    <b>Male Swimwear Or Underwear</b>
    Visuals depicting male person wearing only swimwear or underwear

    <b>Partial Nudity</b>
    Visuals depicting covered up nudity, for example using hands or pose

    <b>Revealing Clothes</b>
    Visuals depicting revealing clothes and poses, such as deep cut dresses
```
<b>Graphic Violence or Gore</b>
Visuals depicting prominent blood or bloody injuries

<b>Physical Violence</b>
Visuals depicting violent physical assault, such as kicking or punching

<b>Weapon Violence</b>
Visuals depicting violence using weapons like firearms or blades, such as shooting

<b>Weapons</b>
Visuals depicting weapons like firearms and blades

<b>Self Injury</b>
Visuals depicting self-inflicted cutting on the body, typically in distinctive patterns using sharp objects

<b>Emaciated Bodies</b>
Visuals depicting extremely malnourished human bodies

<b>Corpses</b>
Visuals depicting human dead bodies

<b>Hanging</b>
Visuals depicting death by hanging
</p>
</short-instructions>
</full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>

### Output

The following is a sample of the output from this element. For details about this output, see Amazon Rekognition DetectModerationLabels API documentation.

```
{
  "AWS/Rekognition/DetectModerationLabels/Image/V3": {
    "ModerationLabels": [
      { name: 'Gore', parentName: 'Violence' },
      { name: 'Corpses', parentName: 'Violence' }
    ]
  }
}
```
Process Data and Evaluate Models

Use Amazon SageMaker Processing to analyze data and evaluate models on the Amazon SageMaker machine learning platform. Amazon SageMaker is a fully managed service that covers the entire machine learning workflow, from preparing your data, to training and deploying the model to make predictions, and monitoring model performance when in production. Data processing tasks such as feature engineering, data validation, model evaluation, and model interpretation are essential steps performed by engineers and data scientists in this machine learning workflow. With Processing you can leverage a simplified, managed experience to run your data processing workloads on the Amazon SageMaker platform or by using the Amazon SageMaker APIs, in the experimentation phase and after code is deployed in production.

You can run an Amazon SageMaker Processing Job to process data from Amazon Simple Storage Service (Amazon S3), and save the processed data back to Amazon S3.

Topics
- Amazon SageMaker Processing Sample Notebooks (p. 189)
- Monitor Amazon SageMaker Processing with CloudWatch Logs and Metrics (p. 190)
- Data Processing and Model Evaluation with Scikit-Learn (p. 190)
- Use Your Own Processing Code (p. 190)

Amazon SageMaker Processing Sample Notebooks

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see scikit-learn Processing. This notebook also shows how to bring your own container to run processing workloads with your own dependencies.

For a sample notebook that shows how to use Processing to do distributed data processing with Spark, see Distributed Processing (Spark).

For instructions on how to create and access Jupyter notebook instances that you can use to run the samples in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After you
have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, choose its Use tab and choose Create copy.

Monitor Amazon SageMaker Processing with CloudWatch Logs and Metrics

Amazon SageMaker Processing provides Amazon CloudWatch logs and metrics to monitor processing jobs. CloudWatch provides CPU, GPU, memory, GPU memory, and disk metrics and event logging. For more information, see Monitor Amazon SageMaker with Amazon CloudWatch (p. 712) and Log Amazon SageMaker Events with Amazon CloudWatch (p. 719).

Data Processing and Model Evaluation with Scikit-Learn

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see scikit-learn Processing. This notebook runs a processing job using SKLearnProcessor to execute a scikit-learn script that you provide that preprocesses data, trains a model using an Amazon SageMaker training job, and then runs a processing job to evaluate the trained model to estimate how the model is expected to perform in production.

The following example shows how to use a SKLearnProcessor to run your own scikit-learn script using a Docker image provided and maintained by Amazon SageMaker, rather than your own Docker image.

```python
from sagemaker.sklearn.preprocessing import SKLearnProcessor
from sagemaker.processing import ProcessingInput, ProcessingOutput

sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                     role=role,
                                     instance_type='ml.m5.xlarge',
                                     instance_count=1)

sklearn_processor.run(code='preprocessing.py',
                      inputs=[ProcessingInput(
                              source='s3://path/to/my/input-data.csv',
                              destination='/opt/ml/processing/input')],
                      outputs=[ProcessingOutput(source='/opt/ml/processing/output/train'),
                               ProcessingOutput(source='/opt/ml/processing/output/validation'),
                               ProcessingOutput(source='/opt/ml/processing/output/test')])
```

Use Your Own Processing Code

You can use your own container to process and install libraries to run your scripts or to build your own processing container. For a formal specification for building a Processing container contract is provided in the Build Your Own Processing Container (p. 191) topic for those who need the details.

Topics
- Run Scripts with Your Own Processing Container (p. 191)
- Build Your Own Processing Container (p. 191)
Run Scripts with Your Own Processing Container

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see the scikit-learn Processing sample.

The following example shows how to use a ScriptProcessor class from the Amazon SageMaker Python SDK to run a Python script with your own image to run a processing job that processes input data, and saves the processed data in Amazon S3.

The notebook shows the general workflow as follows. It includes the code from the steps that use the Amazon SageMaker Python SDK.

1. Write the Dockerfile with the dependencies, build it, and push it to an Amazon Elastic Container Registry (Amazon ECR) repository.
2. Write the preprocessing.py that contains the preprocessing code.
3. Set up the ScriptProcessor to run the script.

```python
from sagemaker.processing import ScriptProcessor, ProcessingInput, ProcessingOutput

script_processor = ScriptProcessor(command=['python3'],
    image_uri='<image_uri>',
    role='<role_arn>',
    instance_count=1,
    instance_type='ml.m5.xlarge')

script_processor.run(code='preprocessing.py',
    inputs=[ProcessingInput(
        source='s3://path/to/my/input-data.csv',
        destination='/opt/ml/processing/input')],
    outputs=[ProcessingOutput(source='/opt/ml/processing/output/train'),
             ProcessingOutput(source='/opt/ml/processing/output/validation'),
             ProcessingOutput(source='/opt/ml/processing/output/test')])
```

4. Run the script.

The same procedure could be used with any other library or system dependencies. More generally, you can use existing Docker images on Amazon SageMaker Processing, including images that you run on other platforms, such Kubernetes.

Build Your Own Processing Container

You can provide SageMaker Processing a Docker image to run your data processing, feature engineering, and model evaluation workloads. You can add your own dependencies to this Docker image, and SageMaker Processing will run this image with your own code and dependencies.

The following example of a Dockerfile builds a container with the Python libraries “scikit-learn” and “pandas” that you can run as a processing job.

```bash
FROM python:3.7-slim-buster

# Install scikit-learn and pandas
RUN pip3 install scikit-learn==0.21.3

# Add a python script and configure Docker to run it
```
ADD processing_script.py /
ENTRYPOINT ["python3", "/processing_script.py"]

After building and pushing this Docker image to an Amazon Elastic Container Registry Amazon ECR) repository and ensuring that your Amazon SageMaker IAM role can pull the image from Amazon ECR, you can run this image on Amazon SageMaker Processing.

How Amazon SageMaker Runs Your Processing Image

Amazon SageMaker Processing runs the container in a similar way as the following command, where AppSpecification.ImageUri is the Amazon ECR image URI that you specify in your CreateProcessingJob API.

```
docker run [AppSpecification.ImageUri]
```

This runs the entrypoint command configured in your Docker image.

You can also override the entrypoint command in the image or give command-line arguments to your entrypoint command using the AppSpecification.ContainerEntrypoint and AppSpecification.ContainerArguments parameters in your CreateProcessingJob request. Specifying these parameters configures Amazon SageMaker Processing to run the container in a similar way as the following command.

```
```

For example, if you specify the ContainerEntrypoint to be ["python3", "-v", "/processing_script.py"] in your CreateProcessingJob request, and ContainerArguments to be ["--data-format", "csv"], Processing runs your container with the command "python3 -v /processing_script.py --data-format csv".

Be aware of the following details when building your processing container:

- Processing decides whether the job completes or fails depending on the exit code of the command run. A processing job completes if all of the processing containers exit successfully with an exit code of 0, and fails if any of the containers exit with a non-zero exit code.

- Processing lets you override the processing container's entrypoint and set command-line arguments just like you can with the Docker API. Docker images can also configure the entrypoint and command-line arguments using the ENTRYPOINT and CMD instructions. The way CreateProcessingJob's ContainerEntrypoint and ContainerArgument parameters configure a Docker image's entrypoint and arguments mirrors how Docker overrides the entrypoint and arguments through the Docker API:
  - If neither ContainerEntrypoint nor ContainerArguments are provided, the default ENTRYPOINT or CMD in the image is used.
  - If ContainerEntrypoint is provided, but not ContainerArguments, the image is run with the given entrypoint, and the ENTRYPOINT and CMD in the image are ignored.
  - If ContainerArguments is provided, but not ContainerEntrypoint, the image is run with the default ENTRYPOINT in the image with the arguments provided.
  - If both ContainerEntrypoint and ContainerArguments are provided, the image is run with the given entrypoint and arguments, and the ENTRYPOINT and CMD in the image are ignored.
  - Use the "exec" form of the ENTRYPOINT instruction in your Dockerfile (ENTRYPOINT ["executable", "param1", "param2"]) instead of the "shell" form (ENTRYPOINT command param1 param2). This lets your processing container receive SIGINT and SIGKILL signals, which Processing uses to stop processing jobs using the StopProcessingJob API.
/opt/ml and all subdirectories are reserved by Amazon SageMaker. When building your processing Docker image, don't place any data required by your processing container in these directories.

- If you plan to use GPU devices, make sure that your containers are nvidia-docker compatible. Only the CUDA toolkit should be included on containers. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see NVIDIA/nvidia-docker.

How Amazon SageMaker Processing Configures Input and Output For Your Processing Container

When you create a processing job using the CreateProcessingJob API, you can specify multiple ProcessingInputs and ProcessingOutputs. values

A ProcessingInput configures an Amazon S3 URI to download data from, and a path in your processing container to download the data to. A ProcessingOutput configures a path in your processing container to upload data from, and where in Amazon S3 to upload that data to. For both ProcessingInput and ProcessingOutput, the path in the processing container must begin with /opt/ml/processing/

For example, you might create a processing job with one ProcessingInput that downloads data from s3://your-data-bucket/path/to/input/csv/data into a path /opt/ml/processing/csv in your processing container, and a ProcessingOutput that uploads data from /opt/ml/processing/processed_csv to s3://your-data-bucket/path/to/output/csv/data. The code running in your processing code might read this data, and write output data to /opt/ml/processing/processed_csv. Processing uploads the data written into this path to the Amazon S3 output location.

How Amazon SageMaker Processing Provides Logs and Metrics for your Processing Container

When your processing container writes to stdout or stderr, Processing saves the output from each processing container and puts the output in Amazon CloudWatch logs. For information about logging, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 719).

Processing also provides CloudWatch metrics for each instance running your processing container. For information about metrics, see the Monitor Amazon SageMaker with Amazon CloudWatch (p. 712).

How Amazon SageMaker Processing Provides Configuration to Your Processing Container

Amazon SageMaker Processing provides configuration to your processing container through environment variables and two JSON files at predefined locations in the processing container, /opt/ml/config/processingjobconfig.json and /opt/ml/config/resourceconfig.json.

Your processing job is started with the environment variables configured using the Environment map in the CreateProcessingJob API request. The file /opt/ml/config/processingjobconfig.json contains information from the CreateProcessingJob request.

The following example shows the format of the file.

```json
{
   "ProcessingJobArn": "<processing_job_arn>",
   "ProcessingJobName": "<processing_job_name>",
   "AppSpecification": {
      "ImageUri": "<image_uri>",
      "ContainerEntrypoint": null,
      "ContainerArguments": null
```


The `/opt/ml/config/resourceconfig.json` file contains information about the hostnames of your processing containers. Use these hostnames when creating or running distributed processing code.

```json
{
  "current_host": "algo-1",
  "hosts": ["algo-1","algo-2","algo-3"]
}
```

Don't use the information in `/etc/hostname` or `/etc/hosts` because it might be inaccurate. Hostname information might not be immediately available to the processing container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

**How You Can Save Metadata Information About Your Processing Job**

Your processing containers can write to a UTF-8 to the file `/opt/ml/output/message` to communicate metadata from the processing container after the processing container exits.
After the processing job enters any terminal status ("Completed", "Stopped", or "Failed"), the "ExitMessage" field in DescribeProcessingJob (p. 1059) will contain the first 1 KB of this file. The DescribeProcessingJob (p. 1059) call returns up to 1 KB of data from your processing containers through the ExitMessage parameter. For example, for failed processing jobs, you can use this field to communicate why the processing container failed.

Do not write sensitive data to this file. If the data in this file is not UTF-8 encoded, the job will fail with a ClientError. If multiple containers exit with "ExitMessage", the "ExitMessage" content from each processing container is concatenated, then truncated to 1 KB.

**How You Can Run Your Processing Container Using the SageMaker Python SDK**

You can use the Amazon SageMaker Python SDK to run your own processing image by using the Processor class. The following example shows how to run your own processing container with one input from Amazon Simple Storage Service (Amazon S3), and one output to Amazon S3.

```python
from sagemaker.processing import Processor, ProcessingInput, ProcessingOutput

processor = Processor(image_uri='<your_ecr_image_uri>',
                       role=role,
                       instance_count=1,
                       instance_type="ml.m5.xlarge")

processor.run(inputs=[[ProcessingInput(
                          source='<s3_uri or local path>',
                          destination='/opt/ml/processing/input_data')],
                       outputs=[[ProcessingOutput(
                          source='/opt/ml/processing/processed_data',
                          destination='<s3_uri>')],
                      ])
```

Instead of building your processing code into your processing image, you can provide a ScriptProcessor with your own image and the command that you want to run, along with the code that you want to run inside that container, as in the Run Scripts with Your Own Processing Container (p. 191) example.

You can also use the scikit-learn image that Processing provides through SKLearnProcessor to run scikit-learn scripts, as in Data Processing and Model Evaluation with Scikit-Learn (p. 190). To learn more about using the Amazon SageMaker Python SDK with Processing containers, see SageMaker Python SDK ReadTheDocs.
Build Models

To build machine learning models in Amazon SageMaker, you have the following options:

- Use one of the built-in algorithms. Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types. For more information, see Use Amazon SageMaker Built-in Algorithms (p. 220).
- Write a custom training script in a machine learning framework that Amazon SageMaker supports, and use one of the pre-built framework containers to run it in Amazon SageMaker. For information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 443).
- Bring your own algorithm or model to train or host in Amazon SageMaker. For information, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 456).
- Use an algorithm that you subscribe to from AWS Marketplace. For information, see Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace (p. 499).

Topics

- Use Amazon SageMaker Notebooks (p. 196)
- Use Amazon SageMaker Notebook Instances (p. 201)
- Choose an Algorithm (p. 220)
- Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace (p. 499)

Use Amazon SageMaker Notebooks

Amazon SageMaker Studio notebooks are preview release and is subject to change.

Amazon SageMaker Studio notebooks are collaborative notebooks that are built into Amazon SageMaker Studio that you can launch quickly. You can access your notebooks without setting up compute instances and file storage so you can get started fast. You pay only for the resources consumed when you run the notebooks. You also have the option of easily switching instance types if you decide you need more or less computing power. This feature is helpful when you are experimenting on a low-cost instance and you decide you want to run a full training session. You can switch to a more powerful instance type to complete the training much more quickly. When training is finished, your instances go away, which saves you money, but the results of your experiments are still available for you to review.

After your organization is set up with Amazon SageMaker Studio, your organization’s data scientists, developers, and other SageMaker users can start working on notebooks within seconds without having to provision any compute resources. Each of your users is automatically assigned their own instance to execute the code of the notebooks, but they can also have multiple instances for different notebooks. Amazon SageMaker Studio notebooks provide persistent storage for your users' notebooks, which enables them to view and share notebooks even if the instances are shut down.

You can share your notebooks with others in your organization, so that they can easily reproduce your results and collaborate while building models and exploring your data. Dependencies for your notebook are included in the notebooks’ environment settings, so sharing is seamless. Sharing publishes a reproducible snapshot of your notebook environments through secure sharable URLs.

To get started, you or your organization's administrator need to complete the Amazon SageMaker Studio setup for the team. There are two modes for authentication: AWS Single Sign-On (AWS SSO) and AWS
Identity and Access Management (IAM). When choosing the appropriate method for your team, you must carefully consider the advantages of each method. Switching to another authentication method after you set up the domain requires a manual migration of the team's notebooks so you want to make sure you choose the appropriate method for your organization at the start. For more information, see the setup documentation.

**Note**
Because Amazon SageMaker Studio Notebooks is in preview, visual elements of Amazon SageMaker Studio might be impacted.

How Are These Notebooks Different from Instance-based Amazon SageMaker Notebooks?

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

When starting a new notebook, we recommend that you use the notebook feature in Amazon SageMaker Studio instead of launching a notebook instance from the console. There are many benefits to using an Amazon SageMaker Studio notebook:

- Starting an Amazon SageMaker Studio notebook is faster than launching an instance-based notebook. Typically, it is 5-10 times faster than instance-based notebooks.
- Notebook sharing is an integrated feature in Amazon SageMaker Studio. Users can generate a shareable link that reproduces not only the notebook code, but also the environment required to execute it, in just a few clicks. You can also share the environment that hosts.
- Amazon SageMaker Studio notebooks come pre-installed with the latest Amazon SageMaker SDK and can be accessed within the studio's IDE, allowing you to build, train, debug, track, and monitor your models.
- As a member of an Amazon SageMaker Studio team, you get a home directory independent of a particular instance. This directory is automatically mounted into all notebook servers and kernels as they're started, so you always have your notebooks and other files. The home directories are in your account in EFS so that you can access them from other services.
- When using AWS SSO, you use your AWS domain credentials and have a unique URL for your team. You never have to interact with the AWS Management Console to run your notebooks.
- Amazon SageMaker Studio Notebooks are equipped with a set of predefined environments to get your organization started on data science projects faster.

When you open a notebook in Amazon SageMaker Studio, the default view is a JupyterLab interface. The primary features are the same, however, so what you can expect to find the typical features of a Jupyter notebook and JupyterLab, you will find here as well.

If you're unfamiliar with JupyterLab features, it is recommended that you take a tour of the JupyterLab user interface features in the JupyterLab documentation.

Usage Metering

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

When you open a notebook, Amazon SageMaker launches a default instance to support execution of your code. Pricing is based on the environment that you select, or if set by your administrator, it's based on the default instance type.
If you run multiple kernels on the same instance, you are billed for the time the instance is running, no matter how many kernels you are running during that time.

For more information about billing and some example scenarios, see Amazon SageMaker Pricing. Note that when you refer to your billing, there may be two types of notebooks listed, Amazon SageMaker Studio Notebook and the instance-based notebook.

Get Started

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

To get started with Amazon SageMaker Studio notebooks you need to set up AWS SSO if you haven't already. You don't have to be in an organization to use AWS SSO. You can sign up as an individual or as an enterprise. The process is similar except enterprises have the option of batch loading their users from their existing AWS SSO organization.

If your organization has already set up a team and you were invited to join the organization or view a specific notebook, you can log in and use the Amazon SageMaker Studio dashboard. From there you can try a tutorial that walks you through all of the features of Amazon SageMaker Studio, including how to use Amazon SageMaker Studio notebooks.

Login

Logging in to an Amazon SageMaker notebook is the same as logging in to Amazon SageMaker Studio. When you receive a shared notebook or if you want to create a new notebook, or run experiments, it is all the same process. Use the console or the link provided to you in an invite email to log in to Amazon SageMaker Studio.

To log in from the console

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Amazon SageMaker Studio.
3. You should see an Amazon SageMaker Studio landing page.
4. If your admin has completed the setup, you will see two sections:
   - **Summary** section that shows your team's domain URL and the status of the domain.
   - In the **Profile** section under the **Summary**, you can access notebooks and manage profiles.

Choose your profile name, and then choose **Open Amazon SageMaker Studio**. In the JupyterLab UI, you can open an existing notebook or create notebooks.

If you can't log in or you didn't receive an invitation in email, you must contact your organization's administrator. For password reset and account recovery instructions, see the SSO guide.

Create a Notebook

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

After you're logged in to Amazon SageMaker Studio, you can create a notebook in the following ways:

- Using the JupyterLab launcher.
Run and Manage Notebooks

- On the **File** menu, choose **New**, and then choose **Notebooks**.

**Environment Options**

Depending on your goals and processing requirements, SageMaker has a collection of pre-built environments.

- Data Science (includes most common data science packages such as NumPy, SciKit Learn, and more)
- Base Python (a plain, vanilla Python environment)
- MXNet CPU optimized
- TensorFlow CPU optimized
- PyTorch CPU optimized

**Use the File Menu to Create a New Notebook**

When you create a new notebook from the File menu, you will be asked to choose a kernel from a dropdown first, before your notebook is created.

**Use the Launcher to Create a New Notebook**

**Selecting an Environment**

When you create a notebook, first you must select the environment. Environments are the combination of software, such as frameworks, and other dependencies that your notebook needs to run. For example, an environment could be TensorFlow with Python 3, Keras with an MXNet backend, or PyTorch and Python 3 customized with additional Python packages.

When you're logged in as an Amazon SageMaker Studio user, you can have one instance of each instance type. Each instance can have only one environment running on it at a time. However, an environment can run multiple kernels or terminal instances.

**Selecting a Kernel**

After selecting an environment, you need to select a kernel. The options are a Python 3 kernel or console, or you can launch a Terminal.

After you choose the kernel, your new notebook launches and opens in a JupyterLab interface. Your notebook will be hosted in a t3.medium instance by default.

**Run and Manage Amazon SageMaker Studio Notebooks**

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

**Topics**

- Change a Notebook's Environment (p. 199)

**Change a Notebook's Environment**

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.
You can change the notebook's kernel in the same way that you change the kernel for any standard Jupyter notebook.

**To change a notebook's environment**

- In the top-right corner of the JupyterLab UI, choose the kernel name.
- Choose from the drop-down list the environment that fits your need. As described previously, you are choosing a kernel that is part of an environment. For example, if you switch from Python 3 (Data Science) to Python 3 (TensorFlow CPU Optimized), you load your notebook in the TensorFlow CPU Optimized environment with a Python 3 kernel.

**Share Amazon SageMaker Studio Notebooks**

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

You can share any of your notebooks with others in your organization. It is a copy, so any changes you make to your notebook aren't reflected in a previous version that you shared. If you want to share your latest version, you must create a new snapshot and then share it.

**To share a notebook**

On the **File** menu, choose **Create Shareable Snapshot**. Or, right-click the notebook and then choose **Create Shareable Snapshot**.

Choose any of the following options:

- **Include github information** - If available, the GitHub repository associated with the notebook. This enables you and your recipient to collaborate and contribute to the same git repository.
- **Include outputs** - You can share a clean notebook, or include the output of your notebook's last run. Check this if you want to share the last run result, especially if the notebook takes long to run.

  **Note**
  If you don’t see some or any of these options, your administrator probably disabled the feature. Contact your administrator.

After selecting your sharing options, you are provided with a URL. You can share this link with users that have access to Studio in your organization. When the user opens the URL, they’re prompted to log in using AWS SSO or IAM. This shared notebook becomes a copy, so changes made by the recipient will not occur in your original notebook.

**Use a Shared Notebook**

You use a shared notebook in the same way you would any notebook that you created yourself. When you click a link to a shared notebook for the first time you will open a read-only version of the notebook. To edit, run and save the shared notebook, choose **Create a Copy** to copy the shared notebook to your personal storage. The notebook launches with the instance and built-in environment that the sender was using when they shared it. Customization to the environment cannot be shared at this moment. You can also inspect the notebook snapshot by choosing **Snapshot Details**.

Important considerations about sharing and authentication:

If you have an active session, you see a read-only view of the notebook until you choose **Create a Copy**.

If you don’t have an active session, you need to log in.
• If you use IAM to login, after you login, select your user profile then choose **Open SageMaker Studio**. Then you need to choose the link you were sent.
• If you use SSO to login, after you login you the shared notebook is opened automatically for you in JupyterLab.

Notebooks are opened based on the sender’s instance type and environments (those are passed as metadata to the notebooks). A new instance of the same type is launched for you when you create a copy of the notebook.

**Use Sample Notebooks**

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

For sample notebooks using Amazon SageMaker, see the [amazon-sagemaker-examples](https://github.com/awsamples/amazon-sagemaker-examples) repository. The provided Amazon SageMaker Studio examples come with the environment settings needed to reproduce the execution.

**View Personal Storage**

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

You can browse the contents of your personal storage by looking at the File Browser in JupyterLab.

**Manage Your Storage Volume**

When you set up Amazon SageMaker Studio, an Amazon Elastic File System (Amazon EFS) file system is created in your AWS account. This volume contains all of the home directories for all of the users in your AWS SSO domain. This is where notebook files and data files are stored. Users can’t directly access each other’s home directories. Don’t delete the Amazon EFS file system. If you do, your AWS SSO domain is longer functional, and all of your users will lose their work.

**Use Amazon SageMaker Notebook Instances**

An **Amazon SageMaker notebook instance** is a fully managed ML compute instance running the Jupyter Notebook App. Amazon SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to Amazon SageMaker hosting, and test or validate your models.

**Topics**

- Create a Notebook Instance (p. 202)
- Access Notebook Instances (p. 205)
- Customize a Notebook Instance (p. 206)
- Use Example Notebooks (p. 208)
- Notebook Instance Software Updates (p. 209)
- Set the Notebook Kernel (p. 210)
- Install External Libraries and Kernels in Notebook Instances (p. 210)
- Associate Git Repositories with Amazon SageMaker Notebook Instances (p. 212)
Create a Notebook Instance

To create a notebook instance, use either the Amazon SageMaker console or the CreateNotebookInstance (p. 913) API.

After receiving the request, Amazon SageMaker does the following:

- **Creates a network interface**—If you choose the optional VPC configuration, it creates the network interface in your VPC. It uses the subnet ID that you provide in the request to determine which Availability Zone to create the subnet in. Amazon SageMaker associates the security group that you provide in the request with the subnet. For more information, see Connect a Notebook Instance to Resources in a VPC (p. 779).

- **Launches an ML compute instance**—Amazon SageMaker launches an ML compute instance in an Amazon SageMaker VPC. Amazon SageMaker performs the configuration tasks that allow it to manage your notebook instance, and if you specified your VPC, it enables traffic between your VPC and the notebook instance.

- **Installs Anaconda packages and libraries for common deep learning platforms**—Amazon SageMaker installs all of the Anaconda packages that are included in the installer. For more information, see Anaconda package list. In addition, Amazon SageMaker installs the TensorFlow and Apache MXNet deep learning libraries.

- **Attaches an ML storage volume**—Amazon SageMaker attaches an ML storage volume to the ML compute instance. You can use the volume to clean up the training dataset or to temporarily store other data to work with. Choose any size between 5 GB and 16384 GB, in 1 GB increments, for the volume. The default is 5 GB. ML storage volumes are encrypted, so Amazon SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

  **Important**
  Only files and data saved within the `/home/ec2-user/SageMaker` folder persist between notebook instance sessions. Files and data that are saved outside this directory are overwritten when the notebook instance stops and restarts.

  **Note**
  Each notebook instance's `/tmp` directory provides a minimum of 10 GB of storage in an instant store. An instance store is temporary, block-level storage that isn't persistent. When the instance is stopped or restarted, Amazon SageMaker deletes the directory's contents. This temporary storage is part of the root volume of the notebook instance.

- **Copies example Jupyter notebooks**—These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

To create a notebook instance:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information:
   a. For **Notebook instance name**, type a name for your notebook instance.
   b. For **Instance type**, choose an instance type for your notebook instance. For a list of supported instance types, see Amazon SageMaker Limits.
c. For Elastic Inference, choose an inference accelerator type to associate with the notebook instance, or choose none. For information about elastic inference, see Use Amazon SageMaker Elastic Inference (EI) (p. 683).

d. For IAM role, choose either an existing IAM role in your account that has the necessary permissions to access Amazon SageMaker resources or Create a new role. If you choose Create a new role, for Create an IAM role:

i. If you want to use S3 buckets other than the one you created in Step 1: Create an Amazon S3 Bucket (p. 25) to store your input data and output, choose them.

   The IAM role automatically has permissions to use any bucket that has sagemaker as part of its name. The AmazonSageMakerFullAccess policy, which Amazon SageMaker attaches to the role, gives the role those permissions.

To give access to other S3 buckets from your notebook instance

- If you're not concerned about users in your AWS account accessing your data, choose Any S3 bucket.
- If your account has sensitive data (such as Human Resources information), restrict access to certain buckets by choosing Specific S3 buckets. You can update the permissions policy attached to the role you are creating later.
- To explicitly control access, restrict access by choosing None. Use bucket and object names and tags as supported by the AmazonSageMakerFullAccess policy. For more information, see AmazonSageMakerFullAccess Policy (p. 770).

ii. Choose Create role.

Amazon SageMaker creates an IAM role named AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmmSS. For example, AmazonSageMaker-ExecutionRole-20171125T090800.

To see the policies that are attached to the role, use the IAM console.

Open the IAM console at https://console.aws.amazon.com/iam/.

You can see that the following policies are attached to the role:

- A trust policy that allows Amazon SageMaker to assume the role.
- The AmazonSageMakerFullAccess AWS managed policy.
- If you gave access to additional S3 bucket(s) when creating this role, the customer managed policy attached to the role. The name of the customer managed policy is AmazonSageMaker-ExecutionPolicy-YYYYMMDDTHHmmSS.

For more information about creating your own IAM role, see Amazon SageMaker Roles (p. 758).

e. For Root access, to enable root access for all notebook instance users, choose Enabled. To disable root access for users, choose Disabled. If you enable root access, all notebook instance users have administrator privileges and can access and edit all files on it.

   Note
   If you disable root access, you will still be able to set up lifecycle configurations, as described later in this procedure.

f. (Optional) Allow access to resources in your Virtual Private Cloud (VPC).

To access resources in your VPC from the notebook instance

i. Choose the VPC and a SubnetId.
For **Security Group**, choose your VPC's default security group. For this exercise and others in this guide, the inbound and outbound rules of the default security group are sufficient.

To allow connecting to a resource in your VPC, ensure that the resource resolves to a private IP address in your VPC. For example, to ensure that an Amazon Redshift DNS name resolves to a private IP address, do one of the following:

- Ensure that the Amazon Redshift cluster is not publicly accessible.
- If the Amazon Redshift cluster is publicly accessible, set the DNS resolution and DNS hostnames VPC parameters to true. For more information, see Managing Clusters in an Amazon Virtual Private Cloud (VPC).

By default, a notebook instance can't connect to on-premises resources or to a peer VPC. You can create a lifecycle configuration that creates an entry in your route table that enables connection to on-premises resources or to a peer VPC. For information, see Understanding Amazon SageMaker notebook instance networking configurations and advanced routing options.

If you allowed access to resources from your VPC, enable direct internet access. For **Direct internet access**, choose Enable. Without internet access, you can't train or host models from notebooks on this notebook instance unless your VPC has a NAT gateway and your security group allows outbound connections. For more information, see Connect a Notebook Instance to Resources in a VPC (p. 779).

(Optional) To use shell scripts that run when you create or start the instance, specify a lifecycle configuration. For information, see Customize a Notebook Instance (p. 206)

(Optional) If you want Amazon SageMaker to use an AWS Key Management Service (AWS KMS) key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.

Specify the size, in GB, of the ML storage volume that is attached to the notebook instance. You can choose a size between 5 GB and 16,384 GB, in 1 GB increments. You can use the volume to clean up the training dataset when you no longer need it or to temporarily store other data to work with.

(Optional) To associate Git repositories with the notebook instance, choose a default repository and up to three additional repositories. For more information, see Associate Git Repositories with Amazon SageMaker Notebook Instances (p. 212).

Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the CreateNotebookInstance (p. 913) API.

When the status of the notebook instance is **InService**, choose **Open Jupyter** next to its name to open the classic Jupyter dashboard, or choose **Open JupyterLab** to open the JupyterLab dashboard. For more information, see Access Notebook Instances (p. 205).

The dashboard provides access to:

- Sample notebooks. Amazon SageMaker provides sample notebooks that contain complete code walkthroughs. These walkthroughs show how to use Amazon SageMaker to perform common machine learning tasks. For more information, see Use Example Notebooks (p. 208).
- The kernels for Jupyter, including those that provide support for Python 2 and 3, Apache MXNet, TensorFlow, PySpark, and R. To create a new notebook and choose a kernel for that notebook, use the **New** menu.

For more information about Jupyter notebooks, see **The Jupyter notebook**.
Access Notebook Instances

To access your Amazon SageMaker notebook instances, choose one of the following options:

- Use the console.

  Choose Notebook instances. The console displays a list of notebook instances in your account. To open a notebook instance with a standard Jupyter interface, choose Open Jupyter for that instance. To open a notebook instance with a JupyterLab interface, choose Open JupyterLab for that instance.

  The console uses your sign-in credentials to send a CreatePresignedNotebookInstanceUrl (p. 924) API request to Amazon SageMaker. Amazon SageMaker returns the URL for your notebook instance, and the console opens the URL in another browser tab and displays the Jupyter notebook dashboard.

  **Note**
  The URL that you get from a call to CreatePresignedNotebookInstanceUrl (p. 924) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the AWS Management Console sign-in page.

- Use the API.

  To get the URL for the notebook instance, call the CreatePresignedNotebookInstanceUrl (p. 924) API and use the URL that the API returns to open the notebook instance.

  Use the Jupyter notebook dashboard to create and manage notebooks and to write code. For more information about Jupyter notebooks, see http://jupyter.org/documentation.html.

Control Root Access to a Notebook Instance

By default, when you create a notebook instance, users that log into that notebook instance have root access. Data science is an iterative process that might require the data scientist to test and use different software tools and packages, so many notebook instance users need to have root access to be able to install these tools and packages. Because users with root access have administrator privileges, users can access and edit all files on a notebook instance with root access enabled.

If you don't want users to have root access to a notebook instance, when you call CreateNotebookInstance (p. 913) or UpdateNotebookInstance (p. 1242) operations, set the RootAccess field to Disabled. You can also disable root access for users when you create or update a notebook instance in the Amazon SageMaker console. For information, see Step 2: Create an Amazon SageMaker Notebook Instance (p. 26).

  **Note**
  Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.
Customize a Notebook Instance

To install packages or sample notebooks on your notebook instance, configure networking and security for it, or otherwise use a shell script to customize it, use a lifecycle configuration. A lifecycle configuration provides shell scripts that run only when you create the notebook instance or whenever you start one. When you create a notebook instance, you can create a new lifecycle configuration and the scripts it uses or apply one that you already have.

The Amazon SageMaker team maintains a public repository of notebook instance lifecycle configurations that address common use cases for customizing notebook instances at https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-configuration-samples.

Note
Each script has a limit of 16384 characters.
The value of the $PATH environment variable that is available to both scripts is /usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin. The working directory, which is the value of the $PWD environment variable, is /
View CloudWatch Logs for notebook instance lifecycle configurations in log group /aws/sagemaker/NotebookInstances in log stream [notebook-instance-name]/[LifecycleConfigHook].
Scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started. To help decrease the run time of scripts, try the following:
- Cut down on necessary steps. For example, limit which conda environments in which to install large packages.
- Run tasks in parallel processes.
- Use the nohup command in your script.

To create a lifecycle configuration
1. For Lifecycle configuration - Optional, choose Create a new lifecycle configuration.
2. For Name, type a name.
3. (Optional) To create a script that runs when you create the notebook and every time you start it, choose Start notebook.
4. In the Start notebook editor, type the script.
5. (Optional) To create a script that runs only once, when you create the notebook, choose Create notebook.
6. In the Create notebook editor, type the script configure networking.
7. Choose Create configuration.

You can see a list of notebook instance lifecycle configurations you previously created by choosing Lifecycle configuration in the Amazon SageMaker console. From there, you can view, edit, delete existing lifecycle configurations. You can create a new notebook instance lifecycle configuration by choosing Create configuration. These notebook instance lifecycle configurations are available when you create a new notebook instance.

Lifecycle Configuration Best Practices

The following are best practices for using lifecycle configurations:
- Lifecycle configurations run as the root user. If your script makes any changes within the /home/ec2-user/SageMaker directory, (for example, installing a package with pip), use the command sudo -u
ec2-user command to run as the ec2-user user. This is the same user that Amazon SageMaker runs as.

- Amazon SageMaker notebook instances use conda environments to implement different kernels for Jupyter notebooks. If you want to install packages that are available to one or more notebook kernels, enclose the commands to install the packages with conda environment commands that activate the conda environment that contains the kernel where you want to install the packages.

For example, if you want to install a package only in for the python3 environment, use the following code:

```bash
#!/bin/bash
sudo -u ec2-user -i <<'EOF'
# This will affect only the Jupyter kernel called "conda_python3".
source activate python3
# Replace myPackage with the name of the package you want to install.
pip install myPackage
# You can also perform "conda install" here as well.
source deactivate
EOF
```

If you want to install a package in all conda environments in the notebook instance, use the following code:

```bash
#!/bin/bash
sudo -u ec2-user -i <<'EOF'
# Note that "base" is special environment name, include it there as well.
for env in base /home/ec2-user/anaconda3/envs/*; do
    source /home/ec2-user/anaconda3/bin/activate $(basename "$env")
    # Installing packages in the Jupyter system environment can affect stability of your SageMaker Notebook Instance. You can remove this check if you'd like to install Jupyter extensions, etc.
    if [ $env = 'JupyterSystemEnv' ]; then
        continue
    fi
    # Replace myPackage with the name of the package you want to install.
pip install --upgrade --quiet myPackage
    # You can also perform "conda install" here as well.
    source /home/ec2-user/anaconda3/bin/deactivate
done
EOF
```

**Important**
All conda environments must be stored in the default environments folder (i.e. /home/ec2-user/anaconda3/envs).

**Important**
When you create or change a script file, we recommend you use Create notebook editor or a text editor that allows for Unix style line breaks. Copying text from a non Linux operating system might include incompatible line breaks and result in an unexpected error.
Use Example Notebooks

Your notebook instance contains example notebooks provided by Amazon SageMaker. The example notebooks contain code that shows how to apply machine learning solutions by using Amazon SageMaker. Notebook instances use the nbexamples Jupyter extension, which enables you to view a read-only version of an example notebook or create a copy of it so that you can modify and run it. For more information about the nbexamples extension, see https://github.com/danielballan/nbexamples.

Note
Example notebooks typically download datasets from the internet. If you disable Amazon SageMaker-provided internet access when you create your notebook instance, example notebooks might not work. For more information, see Connect a Notebook Instance to Resources in a VPC (p. 779).

Use or View Example Notebooks in Jupyter Classic

To view or use the example notebooks in the classic Jupyter view, choose the SageMaker Examples tab.

![](image)

To view a read-only version of an example notebook in the Jupyter classic view, on the SageMaker Examples tab, choose Preview for that notebook. To create a copy of an example notebook in the home directory of your notebook instance, choose Use. In the dialog box, you can change the notebook's name before saving it.

Use or View Example Notebooks in Jupyterlab

To view or use the example notebooks in the Jupyterlab view, choose the examples icon in the left navigation panel.
To view a read-only version of an example notebook, choose the name of the notebook. This opens the notebook as a tab in the main area. To create a copy of an example notebook in the home directory of your notebook instance, choose Create a Copy in the top banner. In the dialog box, type a name for the notebook and then choose CREATE COPY.

For more information about the example notebooks, see the Amazon SageMaker examples GitHub repository.

**Notebook Instance Software Updates**

Amazon SageMaker periodically tests and releases software that is installed on notebook instances. This includes:

- Kernel updates
- Security patches
- AWS SDK updates
- Amazon SageMaker Python SDK updates
- Open source software updates
Set the Notebook Kernel

Amazon SageMaker does not automatically update software on a notebook instance when it is in service. To ensure that you have the most recent software updates, stop and restart your notebook instance, either in the Amazon SageMaker console or by calling StopNotebookInstance (p. 1220) followed by StartNotebookInstance (p. 1208).

You can also manually update software installed on your notebook instance while it is running by using update commands in a terminal or in a notebook.

**Note**

Updating kernels and some packages might depend on whether root access is enabled for the notebook instance. For more information, see Control Root Access to a Notebook Instance (p. 205).

Notebook instances do not notify you if you are running outdated software. You can check the Personal Health Dashboard or the security bulletin at https://aws.amazon.com/security/security-bulletins/ for updates.

Set the Notebook Kernel

Amazon SageMaker provides several kernels for Jupyter that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To set a kernel for a new notebook in the Jupyter notebook dashboard, choose New, and then choose the kernel from the list.

Install External Libraries and Kernels in Notebook Instances

Amazon SageMaker notebook instances come with multiple environments already installed. These environments contain Jupyter kernels and Python packages including: scikit, Pandas, NumPy, TensorFlow, and MXNet. These environments, along with all files in the sample-notebooks folder, are refreshed when you stop and start a notebook instance. You can also install your own environments that contain your choice of packages and kernels. This is typically done using conda install or pip install.

The different Jupyter kernels in Amazon SageMaker notebook instances are separate conda environments. For information about conda environments, see Managing environments in the Conda documentation. If you want to use an external library in a specific kernel, install the library in the environment for that kernel. You can do this either in the terminal or in a notebook cell. The following procedures show how to install Theano so that you can use it in a notebook with a conda_mxnet_p36 kernel.

**To install Theano from a terminal**

1. Open a notebook instance.
2. In the Jupyter dashboard, choose New, and then choose Terminal.
3. In the terminal, type the following commands:
To install Theano from a Jupyter notebook cell

1. Open a notebook instance.
2. In the Jupyter dashboard, choose New, and then choose conda_mxnet_p36.
3. In a cell in the new notebook, type the following command:

   ```
   !pip install theano
   ```

Maintain a Sandboxed Python Environment

Amazon SageMaker periodically updates the Python and dependency versions in the environments installed on a notebook instance when it is stopped and restarted. For more information, see Notebook Instance Software Updates (p. 209). To maintain an isolated Python environment that does not change versions, create a lifecycle configuration that runs each time you start your notebook instance. For information about creating lifecycle configurations, see Customize a Notebook Instance (p. 206).

The following example lifecycle configuration script installs Miniconda on your notebook instance. This allows you to create environments in your notebook instance with specific versions of Python and dependencies that Amazon SageMaker does not update:

```bash
#!/bin/bash
set -e
WORKING_DIR=/home/ec2-user/.myproject
mkdir -p "$WORKING_DIR"

# Install Miniconda to get a separate python and pip
wget https://repo.anaconda.com/miniconda/Miniconda3-4.5.12-Linux-x86_64.sh -O "$WORKING_DIR/miniconda.sh"

# Install Miniconda into the working directory
bash "$WORKING_DIR/miniconda.sh" -b -u -p "$WORKING_DIR/miniconda"

# Install pinned versions of any dependencies
source "$WORKING_DIR/miniconda/bin/activate"
pip install boto3==1.9.86
pip install requests==2.21.0

# Bootstrapping code

# Cleanup
source "$WORKING_DIR/miniconda/bin/deactivate"
rm -rf "$WORKING_DIR/miniconda.sh"
```

You can also add a sandboxed Python installation as a kernel that you can use in a Jupyter notebook by including the following code to the above lifecycle configuration:

```bash
source "$WORKING_DIR/miniconda/bin/activate"
# If required, add this as a kernel
```
pip install ipykernel
python -m ipykernel install --user --name MyProjectEnv --display-name "Python (myprojectenv)"
source "$WORKING_DIR/miniconda/bin/deactivate"

**Associate Git Repositories with Amazon SageMaker Notebook Instances**

Associate Git repositories with your notebook instance to save your notebooks in a source control environment that persists even if you stop or delete your notebook instance. You can associate one default repository and up to three additional repositories with a notebook instance. The repositories can be hosted in AWS CodeCommit, GitHub or on any other Git server. Associating Git repositories with your notebook instance can be useful for:

- **Persistence** - Notebooks in a notebook instance are stored on durable Amazon EBS volumes, but they do not persist beyond the life of your notebook instance. Storing notebooks in a Git repository enables you to store and use notebooks even if you stop or delete your notebook instance.
- **Collaboration** - Peers on a team often work on machine learning projects together. Storing your notebooks in Git repositories allows peers working in different notebook instances to share notebooks and collaborate on them in a source-control environment.
- **Learning** - Many Jupyter notebooks that demonstrate machine learning techniques are available in publicly hosted Git repositories, such as on GitHub. You can associate your notebook instance with a repository to easily load Jupyter notebooks contained in that repository.

There are two ways to associate a Git repository with a notebook instance:

- **Add a Git repository as a resource in your Amazon SageMaker account.** Then, to access the repository, you can specify an AWS Secrets Manager secret that contains credentials. That way, you can access repositories that require authentication.
- **Associate a public Git repository that is not a resource in your account.** If you do this, you cannot specify credentials to access the repository.

**Topics**

- Add a Git Repository to Your Amazon SageMaker Account (p. 212)
- Create a Notebook Instance with an Associated Git Repository (p. 215)
- Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance (p. 216)
- Use Git Repositories in a Notebook Instance (p. 217)

**Add a Git Repository to Your Amazon SageMaker Account**

To manage your GitHub repositories, easily associate them with your notebook instances, and associate credentials for repositories that require authentication, add the repositories as resources in your Amazon SageMaker account. You can view a list of repositories that are stored in your account and details about each repository in the Amazon SageMaker console and by using the API.

You can add Git repositories to your Amazon SageMaker account in the Amazon SageMaker console or by using the AWS CLI.

**Note**

You can use the Amazon SageMaker API `CreateCodeRepository (p. 866)` to add Git repositories to your Amazon SageMaker account, but step-by-step instructions are not provided here.
Add a Git Repository to Your Amazon SageMaker Account (Console)

To add a Git repository as a resource in your Amazon SageMaker account

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Git repositories, then choose Add repository.
3. To add a CodeCommit repository, choose AWS CodeCommit.
   a. To use an existing CodeCommit repository:
      i. Choose Use existing repository.
      ii. For Repository, choose a repository from the list.
      iii. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      iv. Choose Add repository.
   b. To create a new CodeCommit repository:
      i. Choose Create new repository.
      ii. Enter a name for the repository that you can use in both CodeCommit and Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      iii. Choose Create repository.
4. To add a Git repository hosted somewhere other than CodeCommit:
   a. Choose GitHub/Other Git-based repo.
   b. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   c. Enter the URL for the repository.
   d. For Git credentials, choose the credentials to use to authenticate to the repository. This is necessary only if the Git repository is private.
      Note
      Do not provide a user name in the URL. Add the username and password in AWS Secrets Manager as described in the next step.
      d. For Git credentials, choose the credentials to use to authenticate to the repository. This is necessary only if the Git repository is private.
      Note
      If you have two-factor authentication enabled for your Git repository, use a personal access token generated by your Git service provider instead of a password.
   i. To use an existing AWS Secrets Manager secret, choose Use existing secret, and then choose a secret from the list. For information about creating and storing a secret, see Creating a Basic Secret in the AWS Secrets Manager User Guide. The name of the secret you use must contain the string sagemaker.
      Note
      The secret must have a staging label of AWSCURRENT and must be in the following format:
      {"username": UserName, "password": Password}
      For GitHub repositories, we recommend using a personal access token instead of your account password. For information, see https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/.
   ii. To create a new AWS Secrets Manager secret, choose Create secret, enter a name for the secret, and then enter the username and password to use to authenticate to the repository. The name for the secret must contain the string sagemaker.
Note
The IAM role you use to create the secret must have the secretsmanager:GetSecretValue permission in its IAM policy.
The secret must have a staging label of AWSCURRENT and must be in the following format:
{"username": UserName, "password": Password}
For GitHub repositories, we recommend using a personal access token instead of your account password.

iii. To not use any credentials, choose No secret.

e. Choose Create secret.

Add a Git Repository to Your Amazon SageMaker Account (CLI)

Use the create-code-repository AWS CLI command. Specify a name for the repository as the value of the code-repository-name argument. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). Also specify the following:

• The default branch
• The URL of the Git repository

Note
Do not provide a user name in the URL. Add the username and password in AWS Secrets Manager as described in the next step.

• The Amazon Resource Name (ARN) of an AWS Secrets Manager secret that contains the credentials to use to authenticate the repository as the value of the git-config argument

For information about creating and storing a secret, see Creating a Basic Secret in the AWS Secrets Manager User Guide. The following command creates a new repository named MyRepository in your Amazon SageMaker account that points to a Git repository hosted at https://github.com/myprofile/my-repo".

For Linux, OS X, or Unix:

```
aws sagemaker create-code-repository \
   --code-repository-name "MyRepository" \ 
```

For Windows:

```
aws sagemaker create-code-repository ^
   --code-repository-name "MyRepository" ^
   --git-config "{\"Branch\":\"master\", \"RepositoryUrl\" : \"https://github.com/myprofile/my-repo\", \"SecretArn\" : ^
   \"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABc0DE\"}"
```

Note
The secret must have a staging label of AWSCURRENT and must be in the following format:
{"username": UserName, "password": Password}
For GitHub repositories, we recommend using a personal access token instead of your account password.
Create a Notebook Instance with an Associated Git Repository

You can associate Git repositories with a notebook instance when you create the notebook instance by using the AWS Management Console, or the AWS CLI.

**Note**
You can use the Amazon SageMaker API CreateNotebookInstance (p. 913) to associate Git repositories with a notebook instance, but step-by-step instructions are not provided here.

**Note**
If you want to use a CodeCommit repository that is in a different AWS than the notebook instance, set up cross-account access for the repository. For information, see Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance (p. 216).

**Topics**
- Create a Notebook Instance with an Associated Git Repository (Console) (p. 215)
- Create a Notebook Instance with an Associated Git Repository (CLI) (p. 215)

Create a Notebook Instance with an Associated Git Repository (Console)

To create a notebook instance and associate Git repositories in the AWS Management Console

1. Follow the instructions at Step 2: Create an Amazon SageMaker Notebook Instance (p. 26).
2. For **Git repositories**, choose Git repositories to associate with the notebook instance.
   a. For **Default repository**, choose a repository that you want to use as your default repository. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose Add a repository to Amazon SageMaker (opens the Add repository flow in a new window) and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 215). To clone a public repository that is not stored in your account, choose Clone a public Git repository to this notebook instance only, and then specify the URL for that repository.
   b. For **Additional repository 1**, choose a repository that you want to add as an additional directory. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose Add a repository to Amazon SageMaker (opens the Add repository flow in a new window) and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 215). To clone a repository that is not stored in your account, choose Clone a public Git repository to this notebook instance only, and then specify the URL for that repository.

Repeat this step up to three times to add up to three additional repositories to your notebook instance.

Create a Notebook Instance with an Associated Git Repository (CLI)

To create a notebook instance and associate Git repositories by using the AWS CLI, use the `create-notebook-instance` command as follows:

- Specify the repository that you want to use as your default repository as the value of the `default-code-repository` argument. Amazon SageMaker clones this repository as a subdirectory in the
Jupyter startup directory at /home/ec2-user/SageMaker. When you open your notebook instance, it opens in this repository. To use a repository that is stored as a resource in your Amazon SageMaker account, specify the name of the repository as the value of the `default-code-repository` argument. To use a repository that is not stored in your account, specify the URL of the repository as the value of the `default-code-repository` argument.

- Specify up to three additional repositories as the value of the `additional-code-repositories` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at /home/ec2-user/SageMaker, and the repository is excluded from the default repository by adding it to the .git/info/exclude directory of the default repository. To use repositories that are stored as resources in your Amazon SageMaker account, specify the names of the repositories as the value of the `additional-code-repositories` argument. To use repositories that are not stored in your account, specify the URLs of the repositories as the value of the `additional-code-repositories` argument.

For example, the following command creates a notebook instance that has a repository named `MyGitRepo`, that is stored as a resource in your Amazon SageMaker account, as a default repository, and an additional repository that is hosted on GitHub:

```
aws sagemaker create-notebook-instance
    --notebook-instance-name "MyNotebookInstance" \
    --instance-type "ml.t2.medium" \
    --role-arn "arn:aws:iam::012345678901:role/service-role/AmazonSageMaker-ExecutionRole-20181129T121390" \
    --default-code-repository "MyGitRepo" \
    --additional-code-repositories "https://github.com/myprofile/my-other-repo"
```

**Note**

If you use an AWS CodeCommit repository that does not contain "SageMaker" in its name, add the `codecommit:GitPull` and `codecommit:GitPush` permissions to the role that you pass as the `role-arn` argument to the `create-notebook-instance` command. For information about how to add permissions to a role, see Adding and Removing IAM Policies in the AWS Identity and Access Management User Guide.

**Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance**

To associate a CodeCommit repository in a different AWS account with your notebook instance, set up cross-account access for the CodeCommit repository.

**To set up cross-account access for a CodeCommit repository and associate it with a notebook instance:**

1. In the AWS account that contains the CodeCommit repository, create an IAM policy that allows access to the repository from users in the account that contains your notebook instance. For information, see Step 1: Create a Policy for Repository Access in AccountA in the CodeCommit User Guide.

2. In the AWS account that contains the CodeCommit repository, create an IAM role, and attach the policy that you created in the previous step to that role. For information, see Step 2: Create a Role for Repository Access in AccountA in the CodeCommit User Guide.

3. Create a profile in the notebook instance that uses the role that you created in the previous step:
   
   a. Open the notebook instance.
   b. Open a terminal in the notebook instance.
   c. Edit a new profile by typing the following in the terminal:
Associate Git Repositories with Amazon SageMaker Notebook Instances

vi /home/ec2-user/.aws/config
d. Edit the file with the following profile information:

```
[profile CrossAccountAccessProfile]
region = us-west-2
role_arn = arn:aws:iam::CodeCommitAccount:role/CrossAccountRepositoryContributorRole
credential_source=Ec2InstanceMetadata
output = json
```

Where CodeCommitAccount is the account that contains the CodeCommit repository, CrossAccountAccessProfile is the name of the new profile, and CrossAccountRepositoryContributorRole is the name of the role you created in the previous step.

4. On the notebook instance, configure git to use the profile you created in the previous step:

   a. Open the notebook instance.
   b. Open a terminal in the notebook instance.
   c. Edit the Git configuration file typing the following in the terminal:

```
vi /home/ec2-user/.gitconfig
d. Edit the file with the following profile information:

```
[credential]
    helper = !aws codecommit credential-helper --
profile CrossAccountAccessProfile #@
UseHttpPath = true
```

Where CrossAccountAccessProfile is the name of the profile that you created in the previous step.

Use Git Repositories in a Notebook Instance

When you open a notebook instance that has Git repositories associated with it, it opens in the default repository, which is installed in your notebook instance directly under /home/ec2-user/SageMaker. You can open and create notebooks, and you can manually run Git commands in a notebook cell. For example:

```
:git pull origin master
```

To open any of the additional repositories, navigate up one folder. The additional repositories are also installed as directories under /home/ec2-user/SageMaker.

If you open the notebook instance with a JupyterLab interface, the jupyter-git extension is installed and available to use. For information about the jupyter-git extension for JupyterLab, see https://github.com/jupyterlab/jupyterlab-git.

When you open a notebook instance in JupyterLab, you see the git repositories associated with it on the left menu:
You can use the jupyter-git extension to manage git visually, instead of using the command line.
Get Notebook Instance Metadata

When you create a notebook instance, Amazon SageMaker creates a JSON file on the instance at the location /opt/ml/metadata/resource-metadata.json that contains the ResourceName and ResourceArn of the notebook instance. You can access this metadata from anywhere within the notebook instance, including in lifecycle configurations. For information about notebook instance lifecycle configurations, see Customize a Notebook Instance (p. 206).

The resource-metadata.json file has the following structure:

```json
{
    "ResourceArn": "NotebookInstanceArn",
    "ResourceName": "NotebookInstanceName"
}
```

You can use this metadata from within the notebook instance to get other information about the notebook instance. For example, the following commands get the tags associated with the notebook instance:

```bash
NOTEBOOK_ARN=$(jq '.ResourceArn' /opt/ml/metadata/resource-metadata.json --raw-output)
aws sagemaker list-tags --resource-arn $NOTEBOOK_ARN
```

The output looks like the following:

```json
{
    "Tags": [
        {
            "Key": "test",
            "Value": "true"
        }
    ]
}
```

Monitor Jupyter Logs in Amazon CloudWatch Logs

Jupyter logs include important information such as events, metrics, and health information that provide actionable insights when running Amazon SageMaker notebooks. By importing Jupyter logs into CloudWatch Logs, customers can use CloudWatch Logs to detect anomalous behaviors, set alarms, and discover insights to keep the Amazon SageMaker notebooks running more smoothly. You can access the logs even when the Amazon EC2 instance that hosts the notebook is unresponsive, and use the logs to troubleshoot the unresponsive notebook. Sensitive information such as AWS account IDs, secret keys, and authentication tokens in presigned URLs are removed so that customers can share logs without leaking private information.

To view Jupyter logs for a notebook instance:

1. Sign in to the AWS Management Console and open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebook instances.
3. In the list of notebook instances, choose the notebook instance for which you want to view Jupyter logs.
4. Under Monitor on the notebook instance details page, choose View logs.
5. In the CloudWatch console, choose the log stream for your notebook instance. Its name is in the form NotebookInstanceName/jupyter.log.
Choose an Algorithm

Use Amazon SageMaker Built-in Algorithms

A machine learning algorithm uses example data to create a generalized solution (a model) that addresses the business question you are trying to answer. After you create a model using example data, you can use it to answer the same business question for a new set of data. This is also referred to as obtaining inferences.

Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types.

Because you create a model to address a business question, your first step is to understand the problem that you want to solve. Specifically, the format of the answer that you are looking for influences the algorithm that you choose. For example, suppose that you are a bank marketing manager, and that you want to conduct a direct mail campaign to attract new customers. Consider the potential types of answers that you're looking for:

- Answers that fit into discrete categories—For example, answers to these questions:
  - "Based on past customer responses, should I mail this particular customer?" Answers to this question fall into two categories, "yes" or "no." In this case, you use the answer to narrow the recipients of the mail campaign.
  - "Based on past customer segmentation, which segment does this customer fall into?" Answers might fall into categories such as "empty nester," "suburban family," or "urban professional." You could use these segments to decide who should receive the mailing.

For this type of discrete classification problem, Amazon SageMaker provides two algorithms: the Linear Learner Algorithm (p. 328) and the XGBoost Algorithm (p. 422). You set the following hyperparameters to direct these algorithms to produce discrete results:

- For the Linear Learner algorithm, set the predictor_type hyperparameter to binary_classifier.
- For the XGBoost algorithm, set the objective hyperparameter to reg:logistic.

- Answers that are quantitative—Consider this question: "Based on the return on investment (ROI) from past mailings, what is the ROI for mailing this customer?" In this case, you use the ROI to target customers for the mail campaign. For these quantitative analysis problems, you can also use the Linear Learner Algorithm (p. 328) or the XGBoost Algorithm (p. 422) algorithms. You set the following hyperparameters to direct these algorithms to produce quantitative results:
  - For the Linear Learner algorithm, set the predictor_type hyperparameter to regressor.
• For the XGBoost algorithm, set the objective hyperparameter to `reg:linear`.

• Answers in the form of discrete recommendations—Consider this question: "Based on past responses to mailings, what is the recommended content for each customer?" In this case, you are looking for a recommendation on what to mail, not whether to mail, the customer. For this problem, Amazon SageMaker provides the Factorization Machines Algorithm (p. 262) algorithm.

All of the questions in the preceding examples rely on having example data that includes answers. There are times that you don't need, or can't get, example data with answers. This is true for problems whose answers identify groups. For example:

• "I want to group current and prospective customers into 10 groups based on their attributes. How should I group them?" You might choose to send the mailing to customers in the group that has the highest percentage of current customers. That is, prospective customers that most resemble current customers based on the same set of attributes. For this type of question, Amazon SageMaker provides the K-Means Algorithm (p. 307).

• "What are the attributes that differentiate these customers, and what are the values for each customer along those dimensions?" You use these answers to simplify the view of current and prospective customers, and, maybe, to better understand these customer attributes. For this type of question, Amazon SageMaker provides the Principal Component Analysis (PCA) Algorithm (p. 390) algorithm.

In addition to these general-purpose algorithms, Amazon SageMaker provides algorithms that are tailored to specific use cases. These include:

• Image Classification Algorithm (p. 271)—Use this algorithm to classify images. It uses example data with answers (referred to as supervised algorithm).

• Sequence-to-Sequence Algorithm (p. 410)—This supervised algorithm is commonly used for neural machine translation.

• Latent Dirichlet Allocation (LDA) Algorithm (p. 323)—This algorithm is suitable for determining topics in a set of documents. It is an unsupervised algorithm, which means that it doesn't use example data with answers during training.

• Neural Topic Model (NTM) Algorithm (p. 343)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

Topics
• Common Elements of Built-in Algorithms (p. 222)
• BlazingText Algorithm (p. 238)
• DeepAR Forecasting Algorithm (p. 247)
• Factorization Machines Algorithm (p. 262)
• Image Classification Algorithm (p. 271)
Use Built-in Algorithms

- IP Insights Algorithm (p. 297)
- K-Means Algorithm (p. 307)
- K-Nearest Neighbors (k-NN) Algorithm (p. 314)
- Latent Dirichlet Allocation (LDA) Algorithm (p. 323)
- Linear Learner Algorithm (p. 328)
- Neural Topic Model (NTM) Algorithm (p. 343)
- Object2Vec Algorithm (p. 349)
- Object Detection Algorithm (p. 365)
- Principal Component Analysis (PCA) Algorithm (p. 390)
- Random Cut Forest (RCF) Algorithm (p. 394)
- Semantic Segmentation Algorithm (p. 402)
- Sequence-to-Sequence Algorithm (p. 410)
- XGBoost Algorithm (p. 422)

Common Elements of Built-in Algorithms

The following topics provide information common to all of the algorithms provided by Amazon SageMaker.

Topics
- Common Parameters for Built-In Algorithms (p. 222)
- Common Data Formats for Built-in Algorithms (p. 228)
- Instance Types for Built-in Algorithms (p. 236)
- Logs for Built-In Algorithms (p. 237)

Common Parameters for Built-In Algorithms

The following table lists parameters for each of the algorithms provided by Amazon SageMaker.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Channel Name</th>
<th>Training Image and Inference Image Registry Path</th>
<th>Training Input Mode</th>
<th>File Type</th>
<th>Instance Class</th>
<th>Parallelizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlazingText</td>
<td>train</td>
<td><code>&lt;ecr_path&gt;/blazingtext:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>Text file (one sentence per line with space-separated tokens)</td>
<td>GPU (single instance only) or CPU</td>
<td>No</td>
</tr>
<tr>
<td>DeepAR Forecasting</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/forecasting-deepar:&lt;tag&gt;</code></td>
<td>File</td>
<td>JSON Lines or Parquet</td>
<td>GPU or CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Factorization Machines</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/factorization-machines:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf</td>
<td>CPU (GPU for dense data)</td>
<td>Yes</td>
</tr>
<tr>
<td>Image Classification</td>
<td>train and validation,</td>
<td><code>&lt;ecr_path&gt;/image-classification:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO or image</td>
<td>GPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>Channel Name</td>
<td>Training Image and Inference Image Registry Path</td>
<td>Training Input Mode</td>
<td>File Type</td>
<td>Instance Class</td>
<td>Parallelizable</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>---------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>IP Insights</td>
<td>train and (optionally) validation</td>
<td>&lt;ecr_path&gt;/ipinsights:&lt;tag&gt;</td>
<td>File</td>
<td>CSV</td>
<td>CPU or GPU</td>
<td>Yes</td>
</tr>
<tr>
<td>k-means</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/kmeans:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU (single GPU device on one or more instances)</td>
<td>No</td>
</tr>
<tr>
<td>k-nearest-neighbor (k-NN)</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/knn:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU (single GPU device on one or more instances)</td>
<td>Yes</td>
</tr>
<tr>
<td>LDA</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/lda:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU (single instance only)</td>
<td>No</td>
</tr>
<tr>
<td>Linear Learner</td>
<td>train and (optionally) validation, test, or both</td>
<td>&lt;ecr_path&gt;/linear-learner:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Neural Topic Model</td>
<td>train and (optionally) validation, test, or both</td>
<td>&lt;ecr_path&gt;/ntm:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Object2Vec</td>
<td>train and (optionally) validation, test, or both</td>
<td>&lt;ecr_path&gt;/object2vec:&lt;tag&gt;</td>
<td>File</td>
<td>JSON Lines</td>
<td>GPU or CPU (single instance only)</td>
<td>No</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>Channel Name</td>
<td>Training Image and Inference Image Registry Path</td>
<td>Training Input Mode</td>
<td>File Type</td>
<td>Instance Class</td>
<td>Parallelizable</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-------------------------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Object Detection</td>
<td>train and validation, (optionally)</td>
<td>&lt;ecr_path&gt;/object-detection:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO or image files (.jpg or .png)</td>
<td>GPU</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>train_annotation, validation_annotation, and model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCA</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/pca:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Random Cut Forest</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/randomcutforest:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>Semantic Segmentation</td>
<td>train and validation, (optionally)</td>
<td>&lt;ecr_path&gt;/semantic-segmentation:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>image files</td>
<td>GPU (single instance only)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>train_annotation, validation_annotation, and (optionally) label_map and model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq2Seq Modeling</td>
<td>train, validation, and vocab</td>
<td>&lt;ecr_path&gt;/seq2seq:&lt;tag&gt;</td>
<td>File</td>
<td>recordIO-protobuf</td>
<td>GPU (single instance only)</td>
<td>No</td>
</tr>
<tr>
<td>XGBoost</td>
<td>train and (optionally) validation</td>
<td>&lt;ecr_path&gt;/xgboost:&lt;tag&gt;</td>
<td>File</td>
<td>CSV or LibSVM</td>
<td>CPU</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Algorithms that are *parallelizable* can be deployed on multiple compute instances for distributed training. For the **Training Image and Inference Image Registry Path** column, use the :1 version tag to ensure that you are using a stable version of the algorithm. You can reliably host a model trained using an image with the :1 tag on an inference image that has the :1 tag. Using the :latest tag in the registry path provides you with the most up-to-date version of the algorithm, but might cause problems with backward compatibility. Avoid using the :latest tag for production purposes.

For the **Training Image and Inference Image Registry Path** column, depending on algorithm and region use one of the following values for `<ecr_path>`.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>AWS Region</th>
<th>Training Image and Inference Image Registry Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factorization Machines, IP Insights, k-means, k-nearest-neighbor, Linear Learner, Object2Vec, Neural Topic Model, PCA, and Random Cut Forest</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>174872318107.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>382416733822.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>AWS Region</td>
<td>Training Image and Inference Image Registry Path</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>us-east-2</td>
<td>us-east-2</td>
<td>404615174143.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>ap-east-1</td>
<td>286214385809.dkr.ecr.ap-east-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>ap-northeast-1</td>
<td>351501993468.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>ap-northeast-2</td>
<td>835164637446.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>ap-southeast-2</td>
<td>712309505854.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>eu-central-1</td>
<td>664544806723.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>eu-north-1</td>
<td>669576153137.dkr.ecr.eu-north-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>eu-west-1</td>
<td>438346466558.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>eu-west-3</td>
<td>749696950732.dkr.ecr.eu-west-3.amazonaws.com</td>
</tr>
<tr>
<td>me-south-1</td>
<td>me-south-1</td>
<td>249704162688.dkr.ecr.me-south-1.amazonaws.com</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>sa-east-1</td>
<td>855470959533.dkr.ecr.sa-east-1.amazonaws.com</td>
</tr>
<tr>
<td>LDA</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>266724342769.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>766337827248.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>999911452149.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>258307448986.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>293181348795.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>AWS Region</td>
<td>Training Image and Inference Image Registry Path</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BlazingText, Image Classification, Object Detection, Semantic Segmentation, Seq2Seq, and XGBoost (0.72)</td>
<td>ap-southeast-2</td>
<td>297031611018.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-central-1</td>
<td>353608530281.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-1</td>
<td>996678624901.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>433757028032.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>811284229777.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>825641698319.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-east-1</td>
<td>286214385809.dkr.ecr.ap-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>501404015308.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>306986355934.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-2</td>
<td>544295431143.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-central-1</td>
<td>813381260812.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-north-1</td>
<td>669576153137.dkr.ecr.eu-north-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-1</td>
<td>685385470294.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-3</td>
<td>749696950732.dkr.ecr.eu-west-3.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>me-south-1</td>
<td>249704162688.dkr.ecr.me-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>sa-east-1</td>
<td>855470959533.dkr.ecr.sa-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>XGBoost (0.90)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>us-west-1</td>
<td>746614075791.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>246618743249.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>683313688378.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>257758044811.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>AWS Region</td>
<td>Training Image and Inference Image Registry Path</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td></td>
<td>414596584902.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td></td>
<td>354813040037.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td></td>
<td>366743142698.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td></td>
<td>121021644041.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td></td>
<td>783357654285.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td>ap-south-1</td>
<td></td>
<td>720646828776.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-east-1</td>
<td></td>
<td>651117190479.dkr.ecr.ap-east-1.amazonaws.com</td>
</tr>
<tr>
<td>ca-central-1</td>
<td></td>
<td>341280168497.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-central-1</td>
<td></td>
<td>492215442770.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-north-1</td>
<td></td>
<td>662702820516.dkr.ecr.eu-north-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-1</td>
<td></td>
<td>141502667606.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-2</td>
<td></td>
<td>764974769150.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-3</td>
<td></td>
<td>659782779980.dkr.ecr.eu-west-3.amazonaws.com</td>
</tr>
<tr>
<td>me-south-1</td>
<td></td>
<td>801668240914.dkr.ecr.me-south-1.amazonaws.com</td>
</tr>
<tr>
<td>sa-east-1</td>
<td></td>
<td>737474898029.dkr.ecr.sa-east-1.amazonaws.com</td>
</tr>
<tr>
<td>DeepAR Forecasting</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>156387875391.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>522234722520.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>566113047672.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-east-1</td>
<td>286214385809.dkr.ecr.ap-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>633353088612.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>204372634319.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
</tbody>
</table>
### Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>AWS Region</th>
<th>Training Image and Inference Image Registry Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap-southeast-2</td>
<td>514117268639.dkr.ecr.ap-southeast-2.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>eu-north-1</td>
<td>669576153137.dkr.ecr.eu-north-1.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>eu-central-1</td>
<td>495149712605.dkr.ecr.eu-central-1.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>eu-west-1</td>
<td>224300973850.dkr.ecr.eu-west-1.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>eu-west-3</td>
<td>749696950732.dkr.ecr.eu-west-3.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>me-south-1</td>
<td>249704162688.dkr.ecr.me-south-1.amazonaws.com</td>
<td></td>
</tr>
<tr>
<td>sa-east-1</td>
<td>855470959533.dkr.ecr.sa-east-1.amazonaws.com</td>
<td></td>
</tr>
</tbody>
</table>

Use the paths and training input mode as follows:

- To create a training job (with a request to the `CreateTrainingJob` API), specify the Docker Registry path and the training input mode for the training image. You create a training job to train a model using a specific dataset.

- To create a model (with a `CreateModel` request), specify the Docker Registry path for the inference image. Amazon SageMaker launches machine learning compute instances that are based on the endpoint configuration and deploys the model, which includes the artifacts (the result of model training).

### Common Data Formats for Built-in Algorithms

The following topics explain the data formats for the algorithms provided by Amazon SageMaker.

**Topics**

- Common Data Formats for Training (p. 228)
- Common Data Formats for Inference (p. 232)

**Common Data Formats for Training**

To prepare for training, you can preprocess your data using a variety of AWS services, including AWS Glue, Amazon EMR, Amazon Redshift, Amazon Relational Database Service, and Amazon Athena. After preprocessing, publish the data to an Amazon S3 bucket. For training, the data need to go through a series of conversions and transformations, including:

- Training data serialization (handled by you)
- Training data deserialization (handled by the algorithm)
- Training model serialization (handled by the algorithm)
- Trained model deserialization (optional, handled by you)
When using Amazon SageMaker in the training portion of the algorithm, make sure to upload all data at once. If more data is added to that location, a new training call would need to be made to construct a brand new model.

The following table lists supported `ContentType` values:

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/csv; label_size=n</td>
<td>Comma-separated values, where n specifies the number of starting columns in a row that are labels. The default value for n is 1.</td>
</tr>
<tr>
<td>application/x-recordio-protobuf</td>
<td>A protobuf message wrapped in a RecordIO record.</td>
</tr>
</tbody>
</table>

**Training Data Formats**

Many Amazon SageMaker algorithms support training with data in CSV format. To use data in CSV format for training, in the input data channel specification, specify `text/csv` as the `ContentType`. Amazon SageMaker requires that a CSV file doesn't have a header record and that the target variable is in the first column. To run unsupervised learning algorithms that don't have a target, specify the number of label columns in the content type. For example, in this case `'text/csv;label_size=0'`.

Most Amazon SageMaker algorithms work best when you use the optimized protobuf recordIO format for the training data. Using this format allows you to take advantage of Pipe mode when training the algorithms that support it. File mode loads all of your data from Amazon Simple Storage Service (Amazon S3) to the training instance volumes. In Pipe mode, your training job streams data directly from Amazon S3. Streaming can provide faster start times for training jobs and better throughput. With Pipe mode, you also reduce the size of the Amazon Elastic Block Store volumes for your training instances. Pipe mode needs only enough disk space to store your final model artifacts. File mode needs disk space to store both your final model artifacts and your full training dataset. See the AlgorithmSpecification (p. 1274) for additional details on the training input mode. For a summary of the data formats supported by each algorithm, see the documentation for the individual algorithms or this table.

**Note**

For an example that shows how to convert the commonly used numPy array into the protobuf recordIO format, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/factorization_machines_mnist/factorization_machines_mnist.ipynb.

In the protobuf recordIO format, Amazon SageMaker converts each observation in the dataset into a binary representation as a set of 4-byte floats and is then loads it to the protobuf values field. If you are using Python for your data preparation, we strongly recommend that you use these existing transformations. However, if you are using another language, the protobuf definition file below provides the schema that you use to convert your data into Amazon SageMaker protobuf format.

```java
syntax = "proto2";

package aialgs.data;

option java_package = "com.amazonaws.aialgorithms.proto";
option java_outer_classname = "RecordProtos";

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float32Tensor {
  // Each value in the vector. If keys is empty, this is treated as a
  // dense vector.
  repeated float values = 1 [packed = true];
}```
// If key is not empty, the vector is treated as sparse, with
// each key specifying the location of the value in the sparse vector.
repeated uint64 keys = 2 [packed = true];

// An optional shape that allows the vector to represent a matrix.
// For example, if shape = [ 10, 20 ], floor(keys[i] / 20) gives the row,
// and keys[i] % 20 gives the column.
// This also supports n-dimensional tensors.
// Note: If the tensor is sparse, you must specify this value.
repeated uint64 shape = 3 [packed = true];

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float64Tensor {
  // Each value in the vector. If keys is empty, this is treated as a
  // dense vector.
  repeated double values = 1 [packed = true];

  // If this is not empty, the vector is treated as sparse, with
  // each key specifying the location of the value in the sparse vector.
  repeated uint64 keys = 2 [packed = true];

  // An optional shape that allows the vector to represent a matrix.
  // For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
  // and keys[i] % 20 gives the column.
  // This also supports n-dimensional tensors.
  // Note: If the tensor is sparse, you must specify this value.
  repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as 32-bit ints (int32).
message Int32Tensor {
  // Each value in the vector. If keys is empty, this is treated as a
  // dense vector.
  repeated int32 values = 1 [packed = true];

  // If this is not empty, the vector is treated as sparse with
  // each key specifying the location of the value in the sparse vector.
  repeated uint64 keys = 2 [packed = true];

  // An optional shape that allows the vector to represent a matrix.
  // For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
  // and keys[i] % 20 gives the column.
  // This also supports n-dimensional tensors.
  // Note: If the tensor is sparse, you must specify this value.
  repeated uint64 shape = 3 [packed = true];
}

// Support for storing binary data for parsing in other ways (such as JPEG/etc).
// This is an example of another type of value and may not immediately be supported.
message Bytes {
  repeated bytes value = 1;

  // If the content type of the data is known, stores it.
  // This allows for the possibility of using decoders for common formats
  // in the future.
  optional string content_type = 2;
}

message Value {
  oneof value {
    // The numbering assumes the possible use of:
    // - float16, float128
    // - int8, int16, int32
    Float32Tensor float32_tensor = 2;
    Float64Tensor float64_tensor = 3;
  }
}
message Record {
  // Map from the name of the feature to the value.
  // For vectors and libsvm-like datasets,
  // a single feature with the name 'values'
  // should be specified.
  map<string, Value> features = 1;

  // An optional set of labels for this record.
  // Similar to the features field above, the key used for
  // generic scalar / vector labels should be 'values'.
  map<string, Value> label = 2;

  // A unique identifier for this record in the dataset.
  // Whilst not necessary, this allows better
  // debugging where there are data issues.
  // This is not used by the algorithm directly.
  optional string uid = 3;

  // Textual metadata describing the record.
  // This may include JSON-serialized information
  // about the source of the record.
  // This is not used by the algorithm directly.
  optional string metadata = 4;

  // An optional serialized JSON object that allows per-record
  // hyper-parameters/configuration/other information to be set.
  // The meaning/interpretation of this field is defined by
  // the algorithm author and may not be supported.
  // This is used to pass additional inference configuration
  // when batch inference is used (e.g. types of scores to return).
  optional string configuration = 5;
}

After creating the protocol buffer, store it in an Amazon S3 location that Amazon SageMaker can access and that can be passed as part of InputDataConfig in create_training_job.

Note
For all Amazon SageMaker algorithms, the ChannelName in InputDataConfig must be set to train. Some algorithms also support a validation or test input channels. These are typically used to evaluate the model's performance by using a hold-out dataset. Hold-out datasets are not used in the initial training but can be used to further tune the model.

Trained Model Deserialization
Amazon SageMaker models are stored as model.tar.gz in the S3 bucket specified in OutputDataConfig S3OutputPath parameter of the create_training_job call. You can specify most of these model artifacts when creating a hosting model. You can also open and review them in your notebook instance. When model.tar.gz is untarred, it contains model_algo-1, which is a serialized Apache MXNet object. For example, you use the following to load the k-means model into memory and view it:

```python
import mxnet as mx
```
print(mx.ndarray.load('model_algo-1'))

**Common Data Formats for Inference**

Amazon SageMaker algorithms accept and produce several different MIME types for the http payloads used in retrieving online and mini-batch predictions. You can use various AWS services to transform or preprocess records prior to running inference. At a minimum, you need to convert the data for the following:

- Inference request serialization (handled by you)
- Inference request deserialization (handled by the algorithm)
- Inference response serialization (handled by the algorithm)
- Inference response deserialization (handled by you)

**Convert Data for Inference Request Serialization**

Content type options for Amazon SageMaker algorithm inference requests include: text/csv, application/json, and application/x-recordio-protobuf. Algorithms that don't support all of these types can support other types. XGBoost, for example, only supports text/csv from this list, but also supports text/libsvm.

For text/csv the value for the Body argument to `invoke_endpoint` should be a string with commas separating the values for each feature. For example, a record for a model with four features might look like: 1.5, 16.0, 14, 23.0. Any transformations performed on the training data should also be performed on the data before obtaining inference. The order of the features matters, and must remain unchanged.

application/json is significantly more flexible and provides multiple possible formats for developers to use in their applications. At a high level, in JavaScript, the payload might look like:

```javascript
let request = {
  // Instances might contain multiple rows that predictions are sought for.
  "instances": [
    // Request and algorithm specific inference parameters.
    "configuration": {},
    // Data in the specific format required by the algorithm.
    "data": {
      "<field name>": dataElement
    }
  ],
}
```

You have the following options for specifying the `dataElement`:

- **Protocol buffers equivalent**:
  
  ```javascript
  // Has the same format as the protocol buffers implementation described for training.
  let dataElement = {
    "keys": [],
    "values": [],
    "shape": []
  }
  ```

- **Simple numeric vector**:
  
  ```javascript
  // An array containing numeric values is treated as an instance containing a
  ```
// single dense vector.
let dataElement = [1.5, 16.0, 14.0, 23.0]

// It will be converted to the following representation by the SDK.
let converted = {
  "features": {
    "values": dataElement
  }
}

And, for multiple records:

let request = {
  "instances": [
    // First instance.
    {
      "features": [ 1.5, 16.0, 14.0, 23.0 ]
    },
    // Second instance.
    {
      "features": [ -2.0, 100.2, 15.2, 9.2 ]
    }
  ]
}

Convert Data for Inference Response Deserialization

Amazon SageMaker algorithms return JSON in several layouts. At a high level, the structure is:

let response = {
  "predictions": [{
    // Fields in the response object are defined on a per algorithm-basis.
  }]
}

The fields that are included in predictions differ across algorithms. The following are examples of output for the k-means algorithm.

Single-record inference:

let response = {
  "predictions": [{
    "closest_cluster": 5,
    "distance_to_cluster": 36.5
  }]
}

Multi-record inference:

let response = {
  "predictions": [
    // First instance prediction.
    {
      "closest_cluster": 5,
      "distance_to_cluster": 36.5
    },
    // Second instance prediction.
    {
      "closest_cluster": 2,
      "distance_to_cluster": 90.3
    }
  ]
}
Multi-record inference with protobuf input:

```
{
  "features": [],
  "label": {
    "closest_cluster": {
      "values": [ 5.0 ] // e.g. the closest centroid/cluster was 1.0
    },
    "distance_to_cluster": {
      "values": [ 36.5 ]
    }
  },
  "uid": "abc123",
  "metadata": "{"created_at": '2017-06-03' }"
}
```

Amazon SageMaker algorithms also support jsonlines format, where the per-record response content is same as that in JSON format. The multi-record structure is a concatenation of per-record response objects separated by newline characters. The response content for the built-in KMeans algorithm for 2 input data points is:

```
{"distance_to_cluster": 23.40593910217285, "closest_cluster": 0.0}
{"distance_to_cluster": 27.250282287597656, "closest_cluster": 0.0}
```

While running batch transform, it is recommended to use jsonlines response type by setting the Accept field in the CreateTransformJobRequest to application/jsonlines.

**Common Request Formats for All Algorithms**

Most algorithms use several of the following inference request formats.

**JSON Request Format**

Content-type: application/json

**Dense Format**

```
let request = {
  "instances": [
    {
      "features": [1.5, 16.0, 14.0, 23.0]
    }
  ]
}
```

```
let request = {
  "instances": [
    {
      "data": {
        "features": {
          "values": [ 1.5, 16.0, 14.0, 23.0 ]
        }
      }
    }
  ]
}
```
Sparse Format

```json
{
  "instances": [
    {  
      "data": {  
        "features": {
          "keys": [26, 182, 232, 243, 431],
          "shape": [2000],
          "values": [1, 1, 1, 4, 1]
        }
      
    },
    {  
      "data": {  
        "features": {
          "keys": [0, 182, 232, 243, 431],
          "shape": [2000],
          "values": [13, 1, 1, 4, 1]
        }
      
    },
    
  ],
}
```

JSONLINES Request Format

Content-type: application/jsonlines

Dense Format

A single record in dense format can be represented as either:

```json
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

or:

```json
{ "data": { "features": { "values": [1.5, 16.0, 14.0, 23.0] } }
```

Sparse Format

A single record in sparse format is represented as:

```json
{ "data": { "features": { "keys": [26, 182, 232, 243, 431], "shape": [2000], "values": [1, 1, 1, 4, 1] } }
```

Multiple records are represented as a concatenation of the above single-record representations, separated by newline characters:

```json
{ "data": { "features": { "keys": [0, 1, 3], "shape": [4], "values": [1, 4, 1] } }  
{ "data": { "features": { "values": [1.5, 16.0, 14.0, 23.0] } }  
{ "features": [1.5, 16.0, 14.0, 23.0] }  
```

CSV Request Format

Content-type: text/csv;label_size=0

**Note**

CSV support is not available for factorization machines.
**RECORDIO Request Format**

Content-type: application/x-recordio-protobuf

**Use Batch Transform with Build-in Algorithms**

While running batch transform, it’s recommended to use jsonlines response type instead of JSON, if supported by the algorithm. This is accomplished by setting the `Accept` field in the `CreateTransformJobRequest` to `application/jsonlines`.

When you create a transform job, the `SplitType` must be set according to the `ContentType` of the input data. Similarly, depending on the `Accept` field in the `CreateTransformJobRequest`, `AssembleWith` must be set accordingly. Please use the following table to help appropriately set these fields:

<table>
<thead>
<tr>
<th>ContentType</th>
<th>Recommended SplitType</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-recordio-protobuf</td>
<td>RecordIO</td>
</tr>
<tr>
<td>text/csv</td>
<td>Line</td>
</tr>
<tr>
<td>application/jsonlines</td>
<td>Line</td>
</tr>
<tr>
<td>application/json</td>
<td>None</td>
</tr>
<tr>
<td>application/x-image</td>
<td>None</td>
</tr>
<tr>
<td>image/*</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accept</th>
<th>Recommended AssembleWith</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-recordio-protobuf</td>
<td>None</td>
</tr>
<tr>
<td>application/json</td>
<td>None</td>
</tr>
<tr>
<td>application/jsonlines</td>
<td>Line</td>
</tr>
</tbody>
</table>

For more information on response formats for specific algorithms, see the following:

- PCA Response Formats (p. 393)
- Linear Learner Response Formats (p. 341)
- NTM Response Formats (p. 348)
- K-Means Response Formats (p. 314)
- Factorization Machine Response Formats (p. 270)

**Instance Types for Built-in Algorithms**

For training and hosting Amazon SageMaker algorithms, we recommend using the following EC2 instance types:

- ml.m4.xlarge, ml.m4.4xlarge, and ml.m4.10xlarge
- ml.c4.xlarge, ml.c4.2xlarge, and ml.c4.8xlarge
- ml.p2.xlarge, ml.p2.8xlarge, and ml.p2.16xlarge

Most Amazon SageMaker algorithms have been engineered to take advantage of GPU computing for training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective.
Exceptions, such as XGBoost, are noted in this guide. (XGBoost implements an open-source algorithm that has been optimized for CPU computation.)

The size and type of data can have a great effect on which hardware configuration is most effective. When the same model is trained on a recurring basis, initial testing across a spectrum of instance types can discover configurations that are more cost effective in the long run. Additionally, algorithms that train most efficiently on GPUs might not require GPUs for efficient inference. Experiment to determine the most cost effectiveness solution.

For more information on Amazon SageMaker hardware specifications, see Amazon SageMaker ML Instance Types.

Logs for Built-In Algorithms

Amazon SageMaker algorithms produce Amazon CloudWatch logs, which provide detailed information on the training process. To see the logs, in the AWS management console, choose CloudWatch, choose Logs, and then choose the /aws/sagemaker/TrainingJobs log group. Each training job has one log stream per node that it was trained on. The log stream's name begins with the value specified in the TrainingJobName parameter when the job was created.

**Note**
If a job fails and logs do not appear in CloudWatch, it's likely that an error occurred before the start of training. Reasons include specifying the wrong training image or S3 location.

The contents of logs vary by algorithms. However, you can typically find the following information:

- Confirmation of arguments provided at the beginning of the log
- Errors that occurred during training
- Measurement of an algorithms accuracy or numerical performance
- Timings for the algorithm, and any major stages within the algorithm

Common Errors

If a training job fails, some details about the failure are provided by the FailureReason return value in the training job description, as follows:

```python
sage = boto3.client('sagemaker')
sage.describe_training_job(TrainingJobName=job_name)['FailureReason']
```

Others are reported only in the CloudWatch logs. Common errors include the following:

1. Misspecifying a hyperparameter or specifying a hyperparameter that is invalid for the algorithm.

   **From the CloudWatch Log:**

   ```
   [10/16/2017 23:45:17 ERROR 139623806805824 train.py:48]
   Additional properties are not allowed (u'mini_batch_siz' was unexpected)
   ```

2. Specifying an invalid value for a hyperparameter.

   **FailureReason:**

   ```
   AlgorithmError: u'abc' is not valid under any of the given
   schemas\nFailed validating u'oneOf' in
   schema[u'properties'][u'feature_dim']:
   {u'oneOf':
   [{u'pattern': u'^([1-9]\[0-9]*$),
   ```

237
BlazingText Algorithm

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms. The Word2vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc. Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.

The Word2vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is called a word embedding. Words that are semantically similar correspond to vectors that are close together. That way, word embeddings capture the semantic relationships between words.

Many natural language processing (NLP) applications learn word embeddings by training on large collections of documents. These pretrained vector representations provide information about semantics and word distributions that typically improves the generalizability of other models that are later trained on a more limited amount of data. Most implementations of the Word2vec algorithm are optimized for multi-core CPU architectures. This makes it difficult to scale to large datasets.

With the BlazingText algorithm, you can scale to large datasets easily. Similar to Word2vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures. BlazingText's implementation of the supervised multi-class, multi-label text classification algorithm extends the fastText text classifier to use GPU acceleration with custom CUDA kernels. You can train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. And, you achieve performance on par with the state-of-the-art deep learning text classification algorithms.

The Amazon SageMaker BlazingText algorithms provides the following features:

- Accelerated training of the fastText text classifier on multi-core CPUs or a GPU and Word2Vec on GPUs using highly optimized CUDA kernels. For more information, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.
- Enriched Word Vectors with Subword Information by learning vector representations for character n-grams. This approach enables BlazingText to generate meaningful vectors for out-of-vocabulary (OOV) words by representing their vectors as the sum of the character n-gram (subword) vectors.
- A batch_skipgram mode for the Word2Vec algorithm that allows faster training and distributed computation across multiple CPU nodes. The batch_skipgram mode does mini-batching using the Negative Sample Sharing strategy to convert level-1 BLAS operations into level-3 BLAS operations. This efficiently leverages the multiply-add instructions of modern architectures. For more information, see Parallelizing Word2Vec in Shared and Distributed Memory.

To summarize, the following modes are supported by BlazingText on different types instances:
<table>
<thead>
<tr>
<th>Modes</th>
<th>Word2Vec (Unsupervised Learning)</th>
<th>Text Classification (Supervised Learning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single CPU instance</td>
<td>cbow</td>
<td>supervised</td>
</tr>
<tr>
<td></td>
<td>Skip-gram</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Batch Skip-gram</td>
<td></td>
</tr>
<tr>
<td>Single GPU instance (with 1 or more GPUs)</td>
<td>cbow</td>
<td>supervised with one GPU</td>
</tr>
<tr>
<td></td>
<td>Skip-gram</td>
<td></td>
</tr>
<tr>
<td>Multiple CPU instances</td>
<td>Batch Skip-gram</td>
<td>None</td>
</tr>
</tbody>
</table>

For more information about the mathematics behind BlazingText, see [BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs](#).

**Topics**
- Input/Output Interface for the BlazingText Algorithm (p. 239)
- EC2 Instance Recommendation for the BlazingText Algorithm (p. 241)
- BlazingText Sample Notebooks (p. 242)
- BlazingText Hyperparameters (p. 242)
- Tune a BlazingText Model (p. 246)

**Input/Output Interface for the BlazingText Algorithm**

The BlazingText algorithm expects a single preprocessed text file with space-separated tokens. Each line in the file should contain a single sentence. If you need to train on multiple text files, concatenate them into one file and upload the file in the respective channel.

**Training and Validation Data Format**

**Training and Validation Data Format for the Word2Vec Algorithm**

For Word2Vec training, upload the file under the `train` channel. No other channels are supported. The file should contain a training sentence per line.

**Training and Validation Data Format for the Text Classification Algorithm**

For supervised mode, you can train with file mode or with the augmented manifest text format.

**Train with File Mode**

For supervised mode, the training/validation file should contain a training sentence per line along with the labels. Labels are words that are prefixed by the string `__label__`. Here is an example of a training/validation file:

```
__label__4  linux ready for prime time , intel says , despite all the linux hype , the open-source movement has yet to make a huge splash in the desktop market . that may be about to change , thanks to chipmaking giant intel corp .

__label__2  bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly as the indian skippers return to international cricket was short lived .
```

239
Note
The order of labels within the sentence doesn't matter.

Upload the training file under the train channel, and optionally upload the validation file under the validation channel.

Train with Augmented Manifest Text Format

The supervised mode also supports the augmented manifest format, which enables you to do training in pipe mode without needing to create RecordIO files. While using the format, an S3 manifest file needs to be generated that contains the list of sentences and their corresponding labels. The manifest file format should be in JSON Lines format in which each line represents one sample. The sentences are specified using the source tag and the label can be specified using the label tag. Both source and label tags should be provided under the AttributeNames parameter value as specified in the request.

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

Model Artifacts and Inference

Model Artifacts for the Word2Vec Algorithm

For Word2Vec training, the model artifacts consist of vectors.txt, which contains words-to-vectors mapping, and vectors.bin, a binary used by BlazingText for hosting, inference, or both. vectors.txt stores the vectors in a format that is compatible with other tools like Gensim and Spacy. For example, a Gensim user can run the following commands to load the vectors.txt file:

```python
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('vectors.txt', binary=False)
word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
word_vectors.doesnt_match("breakfast cereal dinner lunch".split())
```

If the evaluation parameter is set to True, an additional file, eval.json, is created. This file contains the similarity evaluation results (using Spearman's rank correlation coefficients) on WS-353 dataset. The number of words from the WS-353 dataset that aren't there in the training corpus are reported.

For inference requests, the model accepts a JSON file containing a list of strings and returns a list of vectors. If the word is not found in vocabulary, inference returns a vector of zeros. If subwords is set to True during training, the model is able to generate vectors for out-of-vocabulary (OOV) words.

Sample JSON Request

Mime-type: application/json

```json
{
  "instances": ["word1", "word2", "word3"]
}
```

Model Artifacts for the Text Classification Algorithm

Training with supervised outputs creates a model.bin file that can be consumed by BlazingText hosting. For inference, the BlazingText model accepts a JSON file containing a list of sentences and returns a list of corresponding predicted labels and probability scores. Each sentence is expected to be a string with space-separated tokens, words, or both.
Sample JSON Request

Mime-type: application/json

```json
{
    "instances": ["the movie was excellent", "i did not like the plot ."]
}
```

By default, the server returns only one prediction, the one with the highest probability. For retrieving the top k predictions, you can set k in the configuration, as follows:

```json
{
    "instances": ["the movie was excellent", "i did not like the plot ."],
    "configuration": {"k": 2}
}
```

For BlazingText, the content-type and accept parameters must be equal. For batch transform, they both need to be application/jsonlines. If they differ, the Accept field is ignored. The format for input follows:

```json
content-type: application/jsonlines

{"source": "source_0"}
{"source": "source_1"}

if you need to pass the value of k for top-k, then you can do it in the following way:

```json
{"source": "source_0", "k": 2}
{"source": "source_1", "k": 3}
```

The format for output follows:

```json
accept: application/jsonlines

{"prob": [prob_1], "label": ["__label__1"]}
{"prob": [prob_1], "label": ["__label__1"]}

If you have passed the value of k to be more than 1, then response will be in this format:

```json
{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
```

For both supervised (text classification) and unsupervised (Word2Vec) modes, the binaries (*.bin) produced by BlazingText can be cross-consumed by fastText and vice versa. You can use binaries produced by BlazingText by fastText. Likewise, you can host the model binaries created with fastText using BlazingText.

For more details on dataset formats and model hosting, see the example notebooks Text Classification with the BlazingText Algorithm, FastText Models, and Generating Subword Embeddings with the Word2Vec Algorithm.

EC2 Instance Recommendation for the BlazingText Algorithm

For cbow and skipgram modes, BlazingText supports single CPU and single GPU instances. Both of these modes support learning of subwords embeddings. To achieve the highest speed without compromising accuracy, we recommend that you use an ml.p3.2xlarge instance.

For batch_skipgram mode, BlazingText supports single or multiple CPU instances. When training on multiple instances, set the value of the S3DataDistributionType field of the
BlazingText Sample Notebooks

For a sample notebook that uses the Amazon SageMaker BlazingText algorithm to train and deploy supervised binary and multiclass classification models, see Blazing Text classification on the DBPedia dataset. For instructions for creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After creating and opening a notebook instance, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the Blazing Text are located in the Introduction to Amazon algorithms section. To open a notebook, choose its Use tab, then choose Create copy.

BlazingText Hyperparameters

When you start a training job with a CreateTrainingJob request, you specify a training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The hyperparameters for the BlazingText algorithm depend on which mode you use: Word2Vec (unsupervised) and Text Classification (supervised).

Word2Vec Hyperparameters

The following table lists the hyperparameters for the BlazingText Word2Vec training algorithm provided by Amazon SageMaker.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>The Word2vec architecture used for training. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: batch_skipgram, skipgram, or cbow</td>
</tr>
<tr>
<td>batch_size</td>
<td>The size of each batch when mode is set to batch_skipgram. Set to a number between 10 and 20. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 11</td>
</tr>
<tr>
<td>buckets</td>
<td>The number of hash buckets to use for subwords. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2000000</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of complete passes through the training data. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>evaluation</strong></td>
<td>Whether the trained model is evaluated using the WordSimilarity-353 Test.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: (Boolean) True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: True</td>
</tr>
<tr>
<td><strong>learning_rate</strong></td>
<td>The step size used for parameter updates.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.05</td>
</tr>
<tr>
<td><strong>min_char</strong></td>
<td>The minimum number of characters to use for subwords/character n-grams.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td><strong>min_count</strong></td>
<td>Words that appear less than min_count times are discarded.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td><strong>max_char</strong></td>
<td>The maximum number of characters to use for subwords/character n-grams</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 6</td>
</tr>
<tr>
<td><strong>negative_samples</strong></td>
<td>The number of negative samples for the negative sample sharing strategy.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
---|---
**sampling_threshold** | The threshold for the occurrence of words. Words that appear with higher frequency in the training data are randomly down-sampled.  
  **Optional**  
  Valid values: Positive fraction. The recommended range is (0, 1e-3]  
  Default value: 0.0001

**subwords** | Whether to learn subword embeddings on not.  
  **Optional**  
  Valid values: (Boolean) True or False  
  Default value: False

**vector_dim** | The dimension of the word vectors that the algorithm learns.  
  **Optional**  
  Valid values: Positive integer  
  Default value: 100

**window_size** | The size of the context window. The context window is the number of words surrounding the target word used for training.  
  **Optional**  
  Valid values: Positive integer  
  Default value: 5

---

**Text Classification Hyperparameters**

The following table lists the hyperparameters for the Text Classification training algorithm provided by Amazon SageMaker.

**Note**  
Although some of the parameters are common between the Text Classification and Word2Vec modes, they might have different meanings depending on the context.

| Parameter Name | Description |
---|---|
**mode** | The training mode.  
  **Required**  
  Valid values: supervised |

**buckets** | The number of hash buckets to use for word n-grams.  
  **Optional**  
  Valid values: Positive integer  
  Default value: 2000000 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| early_stopping    | Whether to stop training if validation accuracy doesn't improve after a patience number of epochs.  
                             Optional  
                             Valid values: (Boolean) True or False  
                             Default value: False |
| epochs            | The maximum number of complete passes through the training data.  
                             Optional  
                             Valid values: Positive integer  
                             Default value: 5 |
| learning_rate     | The step size used for parameter updates.  
                             Optional  
                             Valid values: Positive float  
                             Default value: 0.05 |
| min_count         | Words that appear less than min_count times are discarded.  
                             Optional  
                             Valid values: Non-negative integer  
                             Default value: 5 |
| min_epochs        | The minimum number of epochs to train before early stopping logic is invoked.  
                             Optional  
                             Valid values: Positive integer  
                             Default value: 5 |
| patience          | The number of epochs to wait before applying early stopping when no progress is made on the validation set. Used only when early_stopping is True.  
                             Optional  
                             Valid values: Positive integer  
                             Default value: 4 |
### Parameter Name | Description
--- | ---
vector_dim | The dimension of the embedding layer.  
**Optional**  
Valid values: Positive integer  
Default value: 100
word_ngrams | The number of word n-gram features to use.  
**Optional**  
Valid values: Positive integer  
Default value: 2

---

**Tune a BlazingText Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning](p. 555).

**Metrics Computed by the BlazingText Algorithm**

The BlazingText Word2Vec algorithm (*skipgram*, *cbow*, and *batch_skipgram* modes) reports on a single metric during training: `train:mean_rho`. This metric is computed on [WS-353 word similarity datasets](#). When tuning the hyperparameter values for the Word2Vec algorithm, use this metric as the objective.

The BlazingText Text Classification algorithm (*supervised* mode), also reports on a single metric during training: the `validation:accuracy`. When tuning the hyperparameter values for the text classification algorithm, use these metrics as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>train:mean_rho</td>
<td>The mean rho (Spearman's rank correlation coefficient) on <a href="#">WS-353 word similarity datasets</a></td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:accuracy</td>
<td>The classification accuracy on the user-specified validation dataset</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

**Tunable BlazingText Hyperparameters**

**Tunable Hyperparameters for the Word2Vec Algorithm**

Tune an Amazon SageMaker BlazingText Word2Vec model with the following hyperparameters. The hyperparameters that have the greatest impact on Word2Vec objective metrics are: `mode`, `learning_rate`, `window_size`, `vector_dim`, and `negative_samples`. 
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges or Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_size</td>
<td>IntegerParameterRange</td>
<td>[8-32]</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>[5-15]</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.005, MaxValue: 0.01</td>
</tr>
<tr>
<td>min_count</td>
<td>IntegerParameterRange</td>
<td>[0-100]</td>
</tr>
<tr>
<td>mode</td>
<td>CategoricalParameterRange</td>
<td>['batch_skipgram', 'skipgram', 'cbow']</td>
</tr>
<tr>
<td>negative_samples</td>
<td>IntegerParameterRange</td>
<td>[5-25]</td>
</tr>
<tr>
<td>sampling_threshold</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0001, MaxValue: 0.001</td>
</tr>
<tr>
<td>vector_dim</td>
<td>IntegerParameterRange</td>
<td>[32-300]</td>
</tr>
<tr>
<td>window_size</td>
<td>IntegerParameterRange</td>
<td>[1-10]</td>
</tr>
</tbody>
</table>

**Tunable Hyperparameters for the Text Classification Algorithm**

Tune an Amazon SageMaker BlazingText text classification model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges or Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>buckets</td>
<td>IntegerParameterRange</td>
<td>[1000000-10000000]</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>[5-15]</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.005, MaxValue: 0.01</td>
</tr>
<tr>
<td>min_count</td>
<td>IntegerParameterRange</td>
<td>[0-100]</td>
</tr>
<tr>
<td>mode</td>
<td>CategoricalParameterRange</td>
<td>['supervised']</td>
</tr>
<tr>
<td>vector_dim</td>
<td>IntegerParameterRange</td>
<td>[32-300]</td>
</tr>
<tr>
<td>word_ngrams</td>
<td>IntegerParameterRange</td>
<td>[1-3]</td>
</tr>
</tbody>
</table>

**DeepAR Forecasting Algorithm**

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

In many applications, however, you have many similar time series across a set of cross-sectional units. For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly...
over all of the time series. DeepAR takes this approach. When your dataset contains hundreds of related time series, DeepAR outperforms the standard ARIMA and ETS methods. You can also use the trained model to generate forecasts for new time series that are similar to the ones it has been trained on.

The training input for the DeepAR algorithm is one or, preferably, more target time series that have been generated by the same process or similar processes. Based on this input dataset, the algorithm trains a model that learns an approximation of this process/processes and uses it to predict how the target time series evolves. Each target time series can be optionally associated with a vector of static (time-independent) categorical features provided by the `cat` field and a vector of dynamic (time-dependent) time series provided by the `dynamic_feat` field. Amazon SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the `context_length` hyperparameter. To control how far in the future predictions can be made, use the `prediction_length` hyperparameter. For more information, see How the DeepAR Algorithm Works (p. 251).

Topics
- Input/Output Interface for the DeepAR Algorithm (p. 248)
- Best Practices for Using the DeepAR Algorithm (p. 250)
- EC2 Instance Recommendations for the DeepAR Algorithm (p. 251)
- DeepAR Sample Notebooks (p. 251)
- How the DeepAR Algorithm Works (p. 251)
- DeepAR Hyperparameters (p. 254)
- Tune a DeepAR Model (p. 258)
- DeepAR Inference Formats (p. 259)

Input/Output Interface for the DeepAR Algorithm

DeepAR supports two data channels. The required `train` channel describes the training dataset. The optional `test` channel describes a dataset that the algorithm uses to evaluate model accuracy after training. You can provide training and test datasets in JSON Lines format. Files can be in gzip or Parquet file format.

When specifying the paths for the training and test data, you can specify a single file or a directory that contains multiple files, which can be stored in subdirectories. If you specify a directory, DeepAR uses all files in the directory as inputs for the corresponding channel, except those that start with a period (.) and those named `_SUCCESS`. This ensures that you can directly use output folders produced by Spark jobs as input channels for your DeepAR training jobs.

By default, the DeepAR model determines the input format from the file extension (.json, .json.gz, or .parquet) in the specified input path. If the path does not end in one of these extensions, you must explicitly specify the format in the SDK for Python. Use the `content_type` parameter of the `s3_input` class.

The records in your input files should contain the following fields:

- `start`—A string with the format `YYYY-MM-DD HH:MM:SS`. The start timestamp can't contain time zone information.
- `target`—An array of floating-point values or integers that represent the time series. You can encode missing values as `null` literals, or as "NaN" strings in JSON, or as `nan` floating-point values in Parquet.
- `dynamic_feat` (optional)—An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must have the same length as the associated `target` value. Missing values are not supported in the features. For example, if target time series represents the demand of different products, an associated
dynamic_feat might be a boolean time-series which indicates whether a promotion was applied (1) to the particular product or not (0):

```
{"start": ..., "target": [1, 5, 10, 2], "dynamic_feat": [[0, 1, 1, 0]]
```

- cat (optional)—An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive integers. For example, the categorical domain \{R, G, B\} can be encoded as \{0, 1, 2\}. All values from each categorical domain must be represented in the training dataset. That's because the DeepAR algorithm can forecast only for categories that have been observed during training. And, each categorical feature is embedded in a low-dimensional space whose dimensionality is controlled by the embedding_dimension hyperparameter. For more information, see DeepAR Hyperparameters (p. 254).

If you use a JSON file, it must be in JSON Lines format. For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],
"dynamic_feat": [[1.1, 1.2, 0.5, ...]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":
[[1.0, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":
[[1.3, 0.4]]}
```

In this example, each time series has two associated categorical features and one time series features.

For Parquet, you use the same three fields as columns. In addition, "start" can be the datetime type. You can compress Parquet files using gzip (gzip) or the Snappy compression library (snappy).

If the algorithm is trained without cat and dynamic_feat fields, it learns a "global" model, that is a model that is agnostic to the specific identity of the target time series at inference time and is conditioned only on its shape.

If the model is conditioned on the cat and dynamic_feat feature data provided for each time series, the prediction will probably be influenced by the character of time series with the corresponding cat features. For example, if the target time series represents the demand of clothing items, you can associate a two-dimensional cat vector that encodes the type of item (e.g. 0 = shoes, 1 = dress) in the first component and the color of an item (e.g. 0 = red, 1 = blue) in the second component. A sample input would look as follows:

```
{ "start": ..., "target": ..., "cat": [0, 0] } # red shoes
{ "start": ..., "target": ..., "cat": [1, 1] } # blue dress
```

At inference time, you can request predictions for targets with cat values that are combinations of the cat values observed in the training data, for example:

```
{ "start": ..., "target": ..., "cat": [0, 1] } # blue shoes
{ "start": ..., "target": ..., "cat": [1, 0] } # red dress
```

The following guidelines apply to training data:

- The start time and length of the time series can differ. For example, in marketing, products often enter a retail catalog at different dates, so their start dates naturally differ. But all series must have the same frequency, number of categorical features, and number of dynamic features.
- Shuffle the training file with respect to the position of the time series in the file. In other words, the time series should occur in random order in the file.
- Make sure to set the start field correctly. The algorithm uses the start timestamp to derive the internal features.
- If you use categorical features (cat), all time series must have the same number of categorical features. If the dataset contains the cat field, the algorithm uses it and extracts the cardinality of the groups from the dataset. By default, cardinality is "auto". If the dataset contains the cat field, but you don’t want to use it, you can disable it by setting cardinality to "". If a model was trained using a cat feature, you must include it for inference.

- If your dataset contains the dynamic_feat field, the algorithm uses it automatically. All time series have to have the same number of feature time series. The time points in each of the feature time series correspond one-to-one to the time points in the target. In addition, the entry in the dynamic_feat field should have the same length as the target. If the dataset contains the dynamic_feat field, but you don’t want to use it, disable it by setting num_dynamic_feat to "". If the model was trained with the dynamic_feat field, you must provide this field for inference. In addition, each of the features has to have the length of the provided target plus the prediction_length. In other words, you must provide the feature value in the future.

If you specify optional test channel data, the DeepAR algorithm evaluates the trained model with different accuracy metrics. The algorithm calculates the root mean square error (RMSE) over the test data as follows:

$$\text{RMSE} = \sqrt{\frac{1}{nT} \sum_{i,t} (\hat{y}_{i,t} - y_{i,t})^2}$$

$y_{i,t}$ is the true value of time series $i$ at the time $t$. $\hat{y}_{i,t}$ is the mean prediction. The sum is over all $n$ time series in the test set and over the last $T$ time points for each time series, where $T$ corresponds to the forecast horizon. You specify the length of the forecast horizon by setting the prediction_length hyperparameter. For more information, see DeepAR Hyperparameters (p. 254).

In addition, the algorithm evaluates the accuracy of the forecast distribution using weighted quantile loss. For a quantile in the range $[0, 1]$, the weighted quantile loss is defined as follows:

$$w\text{QuantileLoss}[\tau] = \frac{1}{\sum_i |y_{i,t}|} \sum_{i,t} Q_{i,t}^{(\tau)}$$

$$Q_{i,t}^{(\tau)} = \begin{cases} (1 - \tau) |q_{i,t}^{(\tau)} - y_{i,t}| & \text{if } q_{i,t}^{(\tau)} > y_{i,t} \\ \tau |q_{i,t}^{(\tau)} - y_{i,t}| & \text{otherwise} \end{cases}$$

$q_{i,t}^{(\tau)}$ is the $\tau$-quantile of the distribution that the model predicts. To specify which quantiles to calculate loss for, set the test_quantiles hyperparameter. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see DeepAR Hyperparameters (p. 254).

For inference, DeepAR accepts JSON format and the following fields:

- "instances", which includes one or more time series in JSON Lines format
- A name of "configuration", which includes parameters for generating the forecast

For more information, see DeepAR Inference Formats (p. 259).

**Best Practices for Using the DeepAR Algorithm**

When preparing your time series data, follow these best practices to achieve the best results:

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference. Regardless of how you set context_length, don’t break up the time series or provide only a part of it. The model uses data points further back than the value set in context_length for the lagged values feature.

- When tuning a DeepAR model, you can split the dataset to create a training dataset and a test dataset. In a typical evaluation, you would test the model on the same time series used for training, but
on the future `prediction_length` time points that follow immediately after the last time point visible during training. You can create training and test datasets that satisfy this criteria by using the entire dataset (the full length of all time series that are available) as a test set and removing the last `prediction_length` points from each time series for training. During training, the model doesn't see the target values for time points on which it is evaluated during testing. During testing, the algorithm withholds the last `prediction_length` points of each time series in the test set and generates a prediction. Then it compares the forecast with the withheld values. You can create more complex evaluations by repeating time series multiple times in the test set, but cutting them at different endpoints. With this approach, accuracy metrics are averaged over multiple forecasts from different time points. For more information, see Tune a DeepAR Model (p. 258).

- Avoid using very large values (>400) for the `prediction_length` because it makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating your data at a higher frequency. For example, use 5min instead of 1min.
- Because lags are used, a model can look further back in the time series than the value specified for `context_length`. Therefore, you don't need to set this parameter to a large value. We recommend starting with the value that you used for `prediction_length`.
- We recommend training a DeepAR model on as many time series as are available. Although a DeepAR model trained on a single time series might work well, standard forecasting algorithms, such as ARIMA or ETS, might provide more accurate results. The DeepAR algorithm starts to outperform the standard methods when your dataset contains hundreds of related time series. Currently, DeepAR requires that the total number of observations available across all training time series is at least 300.

**EC2 Instance Recommendations for the DeepAR Algorithm**

You can train DeepAR on both GPU and CPU instances and in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, `ml.c4.2xlarge` or `ml.c4.4xlarge`), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves throughput only for larger models (with many cells per layer and many layers) and for large mini-batch sizes (for example, greater than 512).

For inference, DeepAR supports only CPU instances.

Specifying large values for `context_length`, `prediction_length`, `num_cells`, `num_layers`, or `mini_batch_size` can create models that are too large for small instances. In this case, use a larger instance type or reduce the values for these parameters. This problem also frequently occurs when running hyperparameter tuning jobs. In that case, use an instance type large enough for the model tuning job and consider limiting the upper values of the critical parameters to avoid job failures.

**DeepAR Sample Notebooks**

For a sample notebook that shows how to prepare a time series dataset for training the Amazon SageMaker DeepAR algorithm and how to deploy the trained model for performing inferences, see Time series forecasting with DeepAR - Synthetic data as well as DeepAR demo on electricity dataset, which illustrates the advanced features of DeepAR on a real world dataset. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker examples. To open a notebook, choose its **Use** tab, and choose **Create copy**.

**How the DeepAR Algorithm Works**

During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series. You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector
of feature time series and a vector of categorical features. For more information, see Input/Output Interface for the DeepAR Algorithm (p. 248).

For example, the following is an element of a training set indexed by \( i \) which consists of a target time series, \( Z_{i,t} \), and two associated feature time series, \( X_{i,1,t} \) and \( X_{i,2,t} \):

\[
\begin{align*}
Z_{i,t} & \\
X_{i,1,t} & \\
X_{i,2,t} &
\end{align*}
\]

The target time series might contain missing values, which are represented by line breaks in the time series. DeepAR supports only feature time series that are known in the future. This allows you to run "what if?" scenarios. What happens, for example, if I change the price of a product in some way?

Each target time series can also be associated with a number of categorical features. You can use these features to encode which groupings a time series belongs to. Categorical features allow the model to learn typical behavior for groups, which it can use to increase model accuracy. DeepAR implements this by learning an embedding vector for each group that captures the common properties of all time series in the group.

**How Feature Time Series Work in the DeepAR Algorithm**

To facilitate learning time-dependent patterns, such as spikes during weekends, DeepAR automatically creates feature time series based on the frequency of the target time series. For example, DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency. It uses these derived feature time series with the custom feature time series that you provide during training and inference. The following figure shows two of these derived time series features: \( u_{i,1,t} \) represents the hour of the day and \( u_{i,2,t} \) the day of the week:

\[
\begin{align*}
Z_{i,t} & \\
u_{i,1,t} & \\
u_{i,2,t} &
\end{align*}
\]

The DeepAR algorithm automatically generates these feature time series. The following table lists the derived features for the supported basic time frequencies.
<table>
<thead>
<tr>
<th>Frequency of the Time Series</th>
<th>Derived Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minute</td>
<td>minute-of-hour, hour-of-day, day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Hour</td>
<td>hour-of-day, day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Day</td>
<td>day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Week</td>
<td>day-of-month, week-of-year</td>
</tr>
<tr>
<td>Month</td>
<td>month-of-year</td>
</tr>
</tbody>
</table>

DeepAR trains a model by randomly sampling several training examples from each of the time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. The `context_length` hyperparameter controls how far in the past the network can see, and the `prediction_length` hyperparameter controls how far in the future predictions can be made. During training, the algorithm ignores training set elements containing time series that are shorter than a specified prediction length. The following figure represents five samples with context lengths of 12 hours and prediction lengths of 6 hours drawn from element $i$. For brevity, we've omitted the feature time series $x_{i,1,t}$ and $u_{i,2,t}$.

![Diagram of time series](image)

To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. In the example with hourly frequency, for each time index, $t = T$, the model exposes the $z_{i,t}$ values, which occurred approximately one, two, and three days in the past.

![Diagram of lagged values](image)
For inference, the trained model takes as input target time series, which might or might not have been used during training, and forecasts a probability distribution for the next `prediction_length` values. Because DeepAR is trained on the entire dataset, the forecast takes into account patterns learned from similar time series.

For information on the mathematics behind DeepAR, see DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks.

### DeepAR Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>context_length</code></td>
<td>The number of time-points that the model gets to see before making the prediction. The value for this parameter should be about the same as the <code>prediction_length</code>. The model also receives lagged inputs from the target, so <code>context_length</code> can be much smaller than typical seasonalities. For example, a daily time series can have yearly seasonality. The model automatically includes a lag of one year, so the context length can be shorter than a year. The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year. Required Valid values: Positive integer</td>
</tr>
<tr>
<td><code>epochs</code></td>
<td>The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also <code>early_stopping_patience</code>. Typical values range from 10 to 1000. Required Valid values: Positive integer</td>
</tr>
<tr>
<td><code>prediction_length</code></td>
<td>The number of time-steps that the model is trained to predict, also called the forecast horizon. The trained model always generates forecasts with this length. It can't generate longer forecasts. The <code>prediction_length</code> is fixed when a model is trained and it cannot be changed later. Required Valid values: Positive integer</td>
</tr>
</tbody>
</table>
| `time_freq`         | The granularity of the time series in the dataset. Use `time_freq` to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, `5min` specifies a frequency of 5 minutes.  
  - `M`: monthly  
  - `W`: weekly  
  - `D`: daily  
  - `H`: hourly  
  - `min`: every minute |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality</td>
<td>When using the categorical features (cat), cardinality is an array specifying the number of categories (groups) per categorical feature. Set this to auto to infer the cardinality from the data. The auto mode also works when no categorical features are used in the dataset. This is the recommended setting for the parameter. Set cardinality to ignore to force DeepAR to not use categorical features, even if they are present in the data. To perform additional data validation, it is possible to explicitly set this parameter to the actual value. For example, if two categorical features are provided where the first has 2 and the other has 3 possible values, set this to ([2, 3]). For more information on how to use categorical features, see the data-section on the main documentation page of DeepAR.</td>
</tr>
<tr>
<td>dropout_rate</td>
<td>The dropout rate to use during training. The model uses zoneout regularization. For each iteration, a random subset of hidden neurons are not updated. Typical values are less than 0.2.</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>If this parameter is set, training stops when no progress is made within the specified number of epochs. The model that has the lowest loss is returned as the final model.</td>
</tr>
</tbody>
</table>

**Optional**

Valid values: float

Default value: 0.1

Valid values: integer
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedding_dimension</td>
<td>Size of embedding vector learned per categorical feature (same value is used for all categorical features). The DeepAR model can learn group-level time series patterns when a categorical grouping feature is provided. To do this, the model learns an embedding vector of size <code>embedding_dimension</code> for each group, capturing the common properties of all time series in the group. A larger <code>embedding_dimension</code> allows the model to capture more complex patterns. However, because increasing the <code>embedding_dimension</code> increases the number of parameters in the model, more training data is required to accurately learn these parameters. Typical values for this parameter are between 10-100. Optional Valid values: positive integer Default value: 10</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate used in training. Typical values range from 1e-4 to 1e-1. Optional Valid values: float Default value: 1e-3</td>
</tr>
<tr>
<td>likelihood</td>
<td>The model generates a probabilistic forecast, and can provide quantiles of the distribution and return samples. Depending on your data, select an appropriate likelihood (noise model) that is used for uncertainty estimates. The following likelihoods can be selected: • <em>gaussian</em>: Use for real-valued data. • <em>beta</em>: Use for real-valued targets between 0 and 1 inclusive. • <em>negative-binomial</em>: Use for count data (non-negative integers). • <em>student-T</em>: An alternative for real-valued data that works well for bursty data. • <em>deterministic-L1</em>: A loss function that does not estimate uncertainty and only learns a point forecast. Optional Valid values: One of <em>gaussian</em>, <em>beta</em>, <em>negative-binomial</em>, <em>student-T</em>, or <em>deterministic-L1</em>. Default value: <em>student-T</em></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The size of mini-batches used during training. Typical values range from 32 to 512.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 128</td>
</tr>
<tr>
<td>num_cells</td>
<td>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 40</td>
</tr>
<tr>
<td>num_dynamic_feat</td>
<td>The number of dynamic features provided in the data. Set this to <strong>auto</strong> to infer the number of dynamic features from the data. The <strong>auto</strong> mode also works when no dynamic features are used in the dataset. This is the recommended setting for the parameter.</td>
</tr>
<tr>
<td></td>
<td>To force DeepAR to not use dynamic features, even if they are present in the data, set <strong>num_dynamic_feat</strong> to <strong>ignore</strong>.</td>
</tr>
<tr>
<td></td>
<td>To perform additional data validation, it is possible to explicitly set this parameter to the actual integer value. For example, if two dynamic features are provided, set this to 2.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <strong>auto</strong>, <strong>ignore</strong>, positive integer, or empty string</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>auto</strong></td>
</tr>
<tr>
<td>num_eval_samples</td>
<td>The number of samples that are used per time-series when calculating test accuracy metrics. This parameter does not have any influence on the training or the final model. In particular, the model can be queried with a different number of samples. This parameter only affects the reported accuracy scores on the test channel after training. Smaller values result in faster evaluation, but then the evaluation scores are typically worse and more uncertain. When evaluating with higher quantiles, for example 0.95, it may be important to increase the number of evaluation samples.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
</tbody>
</table>

---

257
Tune a DeepAR Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning](p. 555).

Metrics Computed by the DeepAR Algorithm

The DeepAR algorithm reports three metrics, which are computed during training. When tuning a model, choose one of these as the objective. For the objective, use either the forecast accuracy on a provided test channel (recommended) or the training loss. For recommendations for the training/test split for the DeepAR algorithm, see [Best Practices for Using the DeepAR Algorithm](p. 250).

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:RMSE</td>
<td>The root mean square error between the forecast and the actual target computed on the test set.</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:mean_wQuantileLoss</td>
<td>The average overall quantile losses computed on the test set. To control which quantiles are used, set the test_quantiles hyperparameter.</td>
<td>Minimize</td>
</tr>
<tr>
<td>train:final_loss</td>
<td>The training negative log-likelihood loss averaged over the last training epoch for the model.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Tunable Hyperparameters for the DeepAR Algorithm

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact, listed in order from the most to least impactful, on DeepAR objective metrics are: epochs, context_length, mini_batch_size, learning_rate, and num_cells.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_layers</td>
<td>The number of hidden layers in the RNN. Typical values range from 1 to 4.</td>
</tr>
<tr>
<td>test_quantiles</td>
<td>Quantiles for which to calculate quantile loss on the test channel.</td>
</tr>
</tbody>
</table>

Optional
Valid values: positive integer
Default value: 2

Valid values: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
### DeepAR Inference Formats

#### DeepAR JSON Request Formats

Query a trained model by using the model's endpoint. The endpoint takes the following JSON request format.

In the request, the `instances` field corresponds to the time series that should be forecast by the model.

If the model was trained with categories, you must provide a `cat` for each instance. If the model was trained without the `cat` field, it should be omitted.

If the model was trained with a custom feature time series (`dynamic_feat`), you have to provide the same number of `dynamic_feat` values for each instance. Each of them should have a length given by `length(target) + prediction_length`, where the last `prediction_length` values correspond to the time points in the future that will be predicted. If the model was trained without custom feature time series, the field should not be included in the request.

```json
{
    "instances": [
        {
            "start": "2009-11-01 00:00:00",
            "target": [4.0, 10.0, "NaN", 100.0, 113.0],
            "cat": [0, 1],
            "dynamic_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]
        },
        {
            "start": "2012-01-30",
            "target": [1.0],
            "cat": [2, 1],
            "dynamic_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]
        },
        {
            "start": "1999-01-30",
```
The configuration field is optional. `configuration.num_samples` sets the number of sample paths that the model generates to estimate the mean and quantiles. `configuration.output_types` describes the information that will be returned in the request. Valid values are "mean" "quantiles" and "samples". If you specify "quantiles", each of the quantile values in `configuration.quantiles` is returned as a time series. If you specify "samples", the model also returns the raw samples used to calculate the other outputs.

**DeepAR JSON Response Formats**

The following is the format of a response, where [...] are arrays of numbers:

```json
{
    "predictions": [
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        },
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        },
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        }
    ]
}
```

DeepAR has a response timeout of 60 seconds. When passing multiple time series in a single request, the forecasts are generated sequentially. Because the forecast for each time series typically takes about 300 to 1000 milliseconds or longer, depending on the model size, passing too many time series in a single request can cause time outs. It's better to send fewer time series per request and send more requests. Because the DeepAR algorithm uses multiple workers per instance, you can achieve much higher throughput by sending multiple requests in parallel.

By default, DeepAR uses one worker per CPU for inference, if there is sufficient memory per CPU. If the model is large and there isn't enough memory to run a model on each CPU, the number of workers is reduced. The number of workers used for inference can be overwritten using the environment variable
Amazon SageMaker Developer Guide
Use Built-in Algorithms

MODEL_SERVER_WORKERS For example, by setting MODEL_SERVER_WORKERS=1) when calling the
Amazon SageMaker CreateModel (p. 902) API.

Batch Transform with the DeepAR Algorithm
DeepAR forecasting supports getting inferences by using batch transform from data using the JSON
Lines format. In this format, each record is represented on a single line as a JSON object, and lines
are separated by newline characters. The format is identical to the JSON Lines format used for model
training. For information, see Input/Output Interface for the DeepAR Algorithm (p. 248). For example:
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],
"dynamic_feat": [[1.1, 1.2, 0.5, ..]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":
[[1.1, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":
[[1.3, 0.4]]}

Note

When creating the transformation job with CreateTransformJob (p. 939), set the
BatchStrategy value to SingleRecord and set the SplitType value in the
TransformInput (p. 1536) conﬁguration to Line, as the default values currently cause runtime
failures.
Similar to the hosted endpoint inference request format, the cat and the dynamic_feat ﬁelds for each
instance are required if both of the following are true:
• The model is trained on a dataset that contained both the cat and the dynamic_feat ﬁelds.
• The corresponding cardinality and num_dynamic_feat values used in the training job are not set
to "".
Unlike hosted endpoint inference, the conﬁguration ﬁeld is set once for the entire batch
inference job using an environment variable named DEEPAR_INFERENCE_CONFIG. The
value of DEEPAR_INFERENCE_CONFIG can be passed when the model is created by calling
CreateTransformJob (p. 939) API. If DEEPAR_INFERENCE_CONFIG is missing in the container
environment, the inference container uses the following default:
{

}

"num_samples": 100,
"output_types": ["mean", "quantiles"],
"quantiles": ["0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"]

The output is also in JSON Lines format, with one line per prediction, in an order identical to the instance
order in the corresponding input ﬁle. Predictions are encoded as objects identical to the ones returned by
responses in online inference mode. For example:
{ "quantiles": { "0.1": [...], "0.2": [...] }, "samples": [...], "mean": [...] }

Note that in the TransformInput (p. 1536) conﬁguration of the Amazon SageMaker
CreateTransformJob (p. 939) request clients must explicitly set the AssembleWith value to Line, as
the default value None concatenates all JSON objects on the same line.
For example, here is an Amazon SageMaker CreateTransformJob (p. 939) request for a DeepAR job with
a custom DEEPAR_INFERENCE_CONFIG:
{

"BatchStrategy": "SingleRecord",

261


Factorization Machines Algorithm

A factorization machine is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically. For example, in a click prediction system, the factorization machine model can capture click rate patterns observed when ads from a certain ad-category are placed on pages from a certain page-category. Factorization machines are a good choice for tasks dealing with high dimensional sparse datasets, such as click prediction and item recommendation.

Note
The Amazon SageMaker implementation of factorization machines considers only pair-wise (2nd order) interactions between features.

Topics
- Input/Output Interface for the Factorization Machines Algorithm (p. 262)
- EC2 Instance Recommendation for the Factorization Machines Algorithm (p. 263)
- Factorization Machines Sample Notebooks (p. 263)
- How Factorization Machines Work (p. 263)
- Factorization Machines Hyperparameters (p. 264)
- Tune a Factorization Machines Model (p. 269)
- Factorization Machine Response Formats (p. 270)

Input/Output Interface for the Factorization Machines Algorithm

The factorization machine algorithm can be run in either in binary classification mode or regression mode. In each mode, a dataset can be provided to the test channel along with the train channel dataset. The scoring depends on the mode used. In regression mode, the testing dataset is scored using Root Mean Square Error (RMSE). In binary classification mode, the test dataset is scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold =0.5).

For training, the factorization machines algorithm currently supports only the recordIO-protobuf format with Float32 tensors. Because their use case is predominantly on sparse data, CSV is not a good candidate. Both File and Pipe mode training are supported for recordIO-wrapped protobuf.

For inference, factorization machines support the application/json and x-recordio-protobuf formats.

- For the binary classification problem, the algorithm predicts a score and a label. The label is a number and can be either 0 or 1. The score is a number that indicates how strongly the algorithm believes that the label should be 1. The algorithm computes score first and then derives the label from the score value. If the score is greater than or equal to 0.5, the label is 1.
• For the **regression** problem, just a score is returned and it is the predicted value. For example, if Factorization Machines is used to predict a movie rating, score is the predicted rating value.

Please see Factorization Machines Sample Notebooks (p. 263) for more details on training and inference file formats.

**EC2 Instance Recommendation for the Factorization Machines Algorithm**

The Amazon SageMaker Factorization Machines algorithm is highly scalable and can train across distributed instances. We recommend training and inference with CPU instances for both sparse and dense datasets. In some circumstances, training with one or more GPUs on dense data might provide some benefit. Training with GPUs is available only on dense data. Use CPU instances for sparse data.

**Factorization Machines Sample Notebooks**

For a sample notebook that uses the Amazon SageMaker factorization machine learning algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Factorization Machines with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its **Use** tab and select **Create copy**.

**How Factorization Machines Work**

The prediction task for a factorization machine model is to estimate a function $\hat{y}$ from a feature set $x_i$ to a target domain. This domain is real-valued for regression and binary for classification. The factorization machine model is supervised and so has a training dataset $(x_i, y_j)$ available. The advantages this model presents lie in the way it uses a factorized parametrization to capture the pairwise feature interactions. It can be represented mathematically as follows:

$$
\hat{y} = w_0 + \sum_i w_i x_i + \sum_{i, j > i} <v_i, v_j> x_i x_j
$$

The three terms in this equation correspond respectively to the three components of the model:

- The $w_0$ term represents the global bias.
- The $w_i$ linear terms model the strength of the $i^{th}$ variable.
- The $<v_i, v_j>$ factorization terms model the pairwise interaction between the $i^{th}$ and $j^{th}$ variable.

The global bias and linear terms are the same as in a linear model. The pairwise feature interactions are modeled in the third term as the inner product of the corresponding factors learned for each feature. Learned factors can also be considered as embedding vectors for each feature. For example, in a classification task, if a pair of features tends to co-occur more often in positive labeled samples, then the inner product of their factors would be large. In other words, their embedding vectors would be close to each other in cosine similarity. For more information about the factorization machine model, see Factorization Machines.

For regression tasks, the model is trained by minimizing the squared error between the model prediction $\hat{y}_n$ and the target value $y_n$. This is known as the square loss:

$$
L = \frac{1}{N} \sum_n (y_n - \hat{y}_n)^2
$$

For a classification task, the model is trained by minimizing the cross entropy loss, also known as the log loss:
\[ L = \frac{1}{N} \sum \limits_n [y_n \log \hat{p}_n + (1 - y_n) \log (1 - \hat{p}_n)] \]

where:
\[ \hat{p}_n = \frac{1}{1 + e^{-y_n}} \]

For more information about loss functions for classification, see [Loss functions for classification](#).

**Factorization Machines Hyperparameters**

The following table contains the hyperparameters for the factorization machines algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The dimension of the input feature space. This could be very high with sparse input.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer. Suggested value range: [10000, 10000000]</td>
</tr>
<tr>
<td>num_factors</td>
<td>The dimensionality of factorization.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer. Suggested value range: [2, 1000], 64 is usually optimal.</td>
</tr>
<tr>
<td>predictor_type</td>
<td>The type of predictor.</td>
</tr>
<tr>
<td></td>
<td>• <strong>binary_classifier</strong>: For binary classification tasks.</td>
</tr>
<tr>
<td></td>
<td>• <strong>regressor</strong>: For regression tasks.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String: binary_classifier or regressor</td>
</tr>
<tr>
<td>bias_init_method</td>
<td>The initialization method for the bias term:</td>
</tr>
<tr>
<td></td>
<td>• <strong>normal</strong>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by bias_init_sigma.</td>
</tr>
<tr>
<td></td>
<td>• <strong>uniform</strong>: Initializes weights with random values uniformly sampled from a range specified by [-bias_init_scale, +bias_init_scale].</td>
</tr>
<tr>
<td></td>
<td>• <strong>constant</strong>: Initializes the weights to a scalar value specified by bias_init_value.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform, normal, or constant</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bias_init_scale</td>
<td>Range for initialization of the bias term. Takes effect if bias_init_method is set to uniform. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: None</td>
</tr>
<tr>
<td>bias_init_sigma</td>
<td>The standard deviation for initialization of the bias term. Takes effect if bias_init_method is set to normal. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01</td>
</tr>
<tr>
<td>bias_init_value</td>
<td>The initial value of the bias term. Takes effect if bias_init_method is set to constant. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512]. Default value: None</td>
</tr>
<tr>
<td>bias_lr</td>
<td>The learning rate for the bias term. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.1</td>
</tr>
<tr>
<td>bias_wd</td>
<td>The weight decay for the bias term. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval [-clip_gradient, +clip_gradient]. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Default value: None</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of training epochs to run.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>eps</td>
<td>Epsilon parameter to avoid division by 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value: small.</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>factors_init_method</td>
<td>The initialization method for factorization terms:</td>
</tr>
<tr>
<td></td>
<td>• normal Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by factors_init_sigma.</td>
</tr>
<tr>
<td></td>
<td>• uniform: Initializes weights with random values uniformly sampled from a range specified by [-factors_init_scale, +factors_init_scale].</td>
</tr>
<tr>
<td></td>
<td>• constant: Initializes the weights to a scalar value specified by factors_init_value.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform, normal, or constant.</td>
</tr>
<tr>
<td></td>
<td>Default value: normal</td>
</tr>
<tr>
<td>factors_init_scale</td>
<td>The range for initialization of factorization terms. Takes effect if factors_init_method is set to uniform.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>factors_init_sigma</td>
<td>The standard deviation for initialization of factorization terms. Takes effect if factors_init_method is set to normal.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>factors_init_value</td>
<td>The initial value of factorization terms. Takes effect if factors_init_method is set to constant.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Valid values: Float. Suggested value range: [1e-8, 512].</td>
<td></td>
</tr>
<tr>
<td>Default value: None</td>
<td></td>
</tr>
<tr>
<td>factors_lr</td>
<td>The learning rate for factorization terms.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
<td></td>
</tr>
<tr>
<td>Default value: 0.0001</td>
<td></td>
</tr>
<tr>
<td>factors_wd</td>
<td>The weight decay for factorization terms.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
<td></td>
</tr>
<tr>
<td>Default value: 0.0001</td>
<td></td>
</tr>
<tr>
<td>linear_lr</td>
<td>The learning rate for linear terms.</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
<td></td>
</tr>
<tr>
<td>Default value: 0.001</td>
<td></td>
</tr>
<tr>
<td>linear_init_method</td>
<td>The initialization method for linear terms:</td>
</tr>
<tr>
<td>• normal Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by linear_init_sigma.</td>
<td></td>
</tr>
<tr>
<td>• uniform Initializes weights with random values uniformly sampled from a range specified by [-linear_init_scale, +linear_init_scale].</td>
<td></td>
</tr>
<tr>
<td>• constant Initializes the weights to a scalar value specified by linear_init_value.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Valid values: uniform, normal, or constant.</td>
<td></td>
</tr>
<tr>
<td>Default value: normal</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>linear_init_scale</td>
<td>Range for initialization of linear terms. Takes effect if</td>
</tr>
<tr>
<td></td>
<td>linear_init_method is set to uniform.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>linear_init_sigma</td>
<td>The standard deviation for initialization of linear terms. Takes effect</td>
</tr>
<tr>
<td></td>
<td>if linear_init_method is set to normal.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</td>
</tr>
<tr>
<td>linear_init_value</td>
<td>The initial value of linear terms. Takes effect if</td>
</tr>
<tr>
<td></td>
<td>linear_init_method is set to constant.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>linear_wd</td>
<td>The weight decay for linear terms.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The size of mini-batch used for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</td>
</tr>
<tr>
<td>rescale_grad</td>
<td>Gradient rescaling optimizer parameter. If set, multiplies the</td>
</tr>
<tr>
<td></td>
<td>gradient with rescale_grad before updating. Often choose to be 1.0/batch_</td>
</tr>
<tr>
<td></td>
<td>size.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
</tbody>
</table>
Tune a Factorization Machines Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

Metrics Computed by the Factorization Machines Algorithm

The factorization machines algorithm has both binary classification and regression predictor types. The predictor type determines which metric you can use for automatic model tuning. The algorithm reports a test:rmse regressor metric, which is computed during training. When tuning the model for regression tasks, choose this metric as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:rmse</td>
<td>Root Mean Square Error</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

The factorization machines algorithm reports three binary classification metrics, which are computed during training. When tuning the model for binary classification tasks, choose one of these as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:binary_classification_accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:binary_classification_cross_entropy</td>
<td>Cross Entropy</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:binary_f_beta</td>
<td>Beta</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable Factorization Machines Hyperparameters

You can tune the following hyperparameters for the factorization machines algorithm. The initialization parameters that contain the terms bias, linear, and factorization depend on their initialization method. There are three initialization methods: uniform, normal, and constant. These initialization methods are not themselves tunable. The parameters that are tunable are dependent on this choice of the initialization method. For example, if the initialization method is uniform, then only the scale parameters are tunable. Specifically, if bias_init_method==uniform, then bias_init_scale, linear_init_scale, and factors_init_scale are tunable. Similarly, if the initialization method is normal, then only sigma parameters are tunable. If the initialization method is constant, then only value parameters are tunable. These dependencies are listed in the following table.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>bias_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>bias_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
<td>Recommended Ranges</td>
<td>Dependency</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>bias_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>bias_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>epoch</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 1000</td>
<td>None</td>
</tr>
<tr>
<td>factors_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>factors_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>factors_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>factors_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>factors_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>linear_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>linear_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>linear_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>linear_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>linear_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRange</td>
<td>MinValue: 100, MaxValue: 10000</td>
<td>None</td>
</tr>
</tbody>
</table>

**Factorization Machine Response Formats**

**JSON Response Format**

Binary classification

```javascript
let response = {
  "predictions": [
    {
      "score": 0.4,
      "predicted_label": 0
    }
  ]
}
```
Regression

```javascript
let response = {
    "predictions": [
        {
            "score": 0.4
        }
    ]
}
```

**JSONLINES Response Format**

Binary classification

```json
{"score": 0.4, "predicted_label": 0}
```

Regression

```json
{"score": 0.4}
```

**RECORDIO Response Format**

Binary classification

```json
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
                values: [0.4]  # float32
            },
            'predicted_label': {
                keys: [],
                values: [0.0]  # float32
            }
        }
    }
]
```

Regression

```json
[
    Record = {
        features = {},
        label = {
            'score': {
                keys: [],
                values: [0.4]  # float32
            }
        }
    }
]
```

**Image Classification Algorithm**

The Amazon SageMaker image classification algorithm is a supervised learning algorithm that supports multi-label classification. It takes an image as input and outputs one or more labels assigned to that
image. It uses a convolutional neural network (ResNet) that can be trained from scratch or trained using transfer learning when a large number of training images are not available.

The recommended input format for the Amazon SageMaker image classification algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format. Refer to this discussion for a broad overview of efficient data preparation and loading for machine learning systems.

**Note**
To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

For more information on convolutional networks, see:

- Deep residual learning for image recognition Kaiming He, et al., 2016 IEEE Conference on Computer Vision and Pattern Recognition
- ImageNet image database
- Image classification in MXNet

**Topics**

- Input/Output Interface for the Image Classification Algorithm (p. 272)
- EC2 Instance Recommendation for the Image Classification Algorithm (p. 289)
- Image Classification Sample Notebooks (p. 289)
- How Image Classification Works (p. 289)
- Image Classification Hyperparameters (p. 290)
- Tune an Image Classification Model (p. 296)

**Input/Output Interface for the Image Classification Algorithm**

The Amazon SageMaker Image Classification algorithm supports both RecordIO (application/x-recordio) and image (image/png, image/jpeg, and application/x-image) content types for training in file mode and supports RecordIO (application/x-recordio) content type for training in pipe mode. However you can also train in pipe mode using the image files (image/png, image/jpeg, and application/x-image), without creating RecordIO files, by using the augmented manifest format. Distributed training is currently not supported in pipe mode and can only be used in file mode. The algorithm supports image/png, image/jpeg, and application/x-image for inference.

**Train with RecordIO Format**

If you use the RecordIO format for training, specify both train and validation channels as values for the InputDataConfig parameter of the CreateTrainingJob (p. 931) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to application/x-recordio.

**Train with Image Format**

If you use the Image format for training, specify train, validation, train_lst, and validation_lst channels as values for the InputDataConfig parameter of the CreateTrainingJob (p. 931) request. Specify the individual image data (.jpg or .png files) for the train and validation channels. Specify one .lst file in each of the train_lst and validation_lst channels. Set the content type for all four channels to application/x-image.

**Note**
Amazon SageMaker reads the training and validation data separately from different channels, so you must store the training and validation data in different folders.

A .lst file is a tab-separated file with three columns that contains a list of image files. The first column specifies the image index, the second column specifies the class label index for the image, and the third
column specifies the relative path of the image file. The image index in the first column must be unique across all of the images. The set of class label indices are numbered successively and the numbering should start with 0. For example, 0 for the cat class, 1 for the dog class, and so on for additional classes.

The following is an example of a .lst file:

<table>
<thead>
<tr>
<th>Image Index</th>
<th>Class Index</th>
<th>Image Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>your_image_directory/train_img_dog1.jpg</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>your_image_directory/train_img_cat1.jpg</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>your_image_directory/train_img_dog2.jpg</td>
</tr>
</tbody>
</table>

For example, if your training images are stored in s3://<your_bucket>/train/class_dog, s3://<your_bucket>/train/class_cat, and so on, specify the path for your train channel as s3://<your_bucket>/train, which is the top-level directory for your data. In the .lst file, specify the relative path for an individual file named train_image_dog1.jpg in the class_dog class directory as class_dog/train_image_dog1.jpg. You can also store all your image files under one subdirectory inside the train directory. In that case, use that subdirectory for the relative path. For example, s3://<your_bucket>/train/your_image_directory.

**Train with Augmented Manifest Image Format**

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the InputDataConfig parameter of the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.
- **HyperParameters** - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- **InputDataConfig** - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- **EnableManagedSpotTraining** - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- **StoppingCondition** - To help cap training costs, use MaxRuntimeInSeconds to set a time limit for training. Use MaxWaitTimeInSeconds to specify how long you are willing to wait for a managed spot training job to complete.
For more information about Amazon SageMaker, see How It Works.

Request Syntax

```json
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "EnableSageMakerMetricsTimeSeries": boolean,
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "DebugHookConfig": {
        "CollectionConfigurations": [
            {
                "CollectionName": "string",
                "CollectionParameters": {
                    "string": "string"
                }
            }
        ],
        "HookParameters": {
            "string": "string"
        },
        "LocalPath": "string",
        "S3OutputPath": "string"
    },
    "DebugRuleConfigurations": [
        {
            "InstanceType": "string",
            "LocalPath": "string",
            "RuleConfigurationName": "string",
            "RuleEvaluatorImage": "string",
            "RuleParameters": {
                "string": "string"
            },
            "S3OutputPath": "string",
            "VolumeSizeInGB": number
        }
    ],
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableManagedSpotTraining": boolean,
    "EnableNetworkIsolation": boolean,
    "ExperimentConfig": {
        "ExperimentName": "string",
        "TrialComponentDisplayName": "string",
        "TrialName": "string"
    },
    "HyperParameters": {
        "string": "string"
    },
    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string"
        }
    ]
}
```
"DataSource": {
  "FileSystemDataSource": {
    "DirectoryPath": "string",
    "FileSystemAccessMode": "string",
    "FileSystemId": "string",
    "FileSystemType": "string"
  },
  "S3DataSource": {
    "AttributeNames": [ "string" ],
    "S3DataDistributionType": "string",
    "S3DataType": "string",
    "S3Uri": "string"
  }
},
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
  "Seed": number
}
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"Tags": [ {
    "Key": "string",
    "Value": "string"
} ],
"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},
"TrainingJobName": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}
about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 1274) object

Required: Yes

**CheckpointConfig (p. 931)**

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

Required: No

**DebugHookConfig (p. 931)**

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

Required: No

**DebugRuleConfigurations (p. 931)**

Configuration information for debugging rules.

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

**EnableInterContainerTrafficEncryption (p. 931)**

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

**EnableManagedSpotTraining (p. 931)**

To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. This option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean
Required: No

**EnableNetworkIsolation (p. 931)**

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

Required: No

**ExperimentConfig (p. 931)**

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

**HyperParameters (p. 931)**

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: .*

Value Length Constraints: Maximum length of 256.

Value Pattern: .*

Required: No

**InputDataConfig (p. 931)**

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, **training_data** and **validation_data**. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects
Array Members: Minimum number of 1 item. Maximum number of 20 items.
Required: No

**OutputDataConfig (p. 931)**

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object
Required: Yes

**ResourceConfig (p. 931)**

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the `TrainingInputMode` in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object
Required: Yes

**RoleArn (p. 931)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**
To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String

Pattern: `^arn:aws[a-zA-Z\-]+:iam::\d{12}:role/?[a-zA-Z0-9+=,.@\-_\/]++$`
Required: Yes

**StoppingCondition (p. 931)**

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object
Required: Yes

**Tags (p. 931)**

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**TensorBoardOutputConfig (p. 931)**

Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

**TrainingJobName (p. 931)**

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

**VpcConfig (p. 931)**

A VpcConfig (p. 1577) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

**Response Syntax**

```json
{
  "TrainingJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrainingJobArn (p. 937)**

The Amazon Resource Name (ARN) of the training job.
Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:training-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p. ) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the 'source-ref' tag that points to the S3 location of the image. The annotations are provided under the "AttributeNames" parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.
If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.

- **HyperParameters** - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

- **InputDataConfig** - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.

- **OutputDataConfig** - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.

- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

- **EnableManagedSpotTraining** - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.

- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.

- **StoppingCondition** - To help cap training costs, use **MaxRuntimeInSeconds** to set a time limit for training. Use **MaxWaitTimeInSeconds** to specify how long you are willing to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

**Request Syntax**

```json
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "EnableSageMakerMetricsTimeSeries": boolean,
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "DebugHookConfig": {
        "CollectionConfigurations": [
            {
                "CollectionName": "string",
                "CollectionParameters": {
                    "string": "string"
                }
            }
        ]
    }
}
```


```
}]

"HookParameters": { 

"string" : "string"
},

"LocalPath": "string",

"S3OutputPath": "string"
},

"DebugRuleConfigurations": [ 

{

"InstanceType": "string",

"LocalPath": "string",

"RuleConfigurationName": "string",

"RuleEvaluatorImage": "string",

"RuleParameters": { 

"string" : "string"
 },

"S3OutputPath": "string",

"VolumeSizeInGB": number
}
],

"EnableInterContainerTrafficEncryption": boolean,

"EnableManagedSpotTraining": boolean,

"EnableNetworkIsolation": boolean,

"ExperimentConfig": { 

"ExperimentName": "string",

"TrialComponentDisplayName": "string",

"TrialName": "string"
},

"HyperParameters": { 

"string" : "string"
}
```
"VolumeKmsKeyId": "string",

"VolumeSizeInGB": number
}

"RoleArn": "string",

"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},

"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
],

"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},

"TrainingJobName": "string",

"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 931)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 1274) object
**DebugRuleConfigurations (p. 931)**

Configuration information for debugging rules.

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

**EnableInterContainerTrafficEncryption (p. 931)**

To encrypt all communications between ML compute instances in distributed training, choose `true`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

**EnableManagedSpotTraining (p. 931)**

To train models using managed spot training, choose `true`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. This option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

**EnableNetworkIsolation (p. 931)**

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

Required: No

**ExperimentConfig (p. 931)**

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

**HyperParameters (p. 931)**

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: . *

Value Length Constraints: Maximum length of 256.

Value Pattern: . *

Required: No

**InputDataConfig (p. 931)**

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

**OutputDataConfig (p. 931)**

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

Required: Yes

**ResourceConfig (p. 931)**

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object

Required: Yes

**InputDataConfig (p. 931)**

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

**OutputDataConfig (p. 931)**

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

Required: Yes

**ResourceConfig (p. 931)**

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object

Required: Yes
**RoleArn (p. 931)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9\+=,.@_-/]+$`

Required: Yes

**StoppingCondition (p. 931)**

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object

Required: Yes

**Tags (p. 931)**

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**TensorBoardOutputConfig (p. 931)**

Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

**TrainingJobName (p. 931)**

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

VpcConfig (p. 931)

A VpcConfig (p. 1577) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

Response Syntax

```json
{
  "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 937)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p. ) request. It can also contain additional metadata under the metadata tag, but these are ignored by the algorithm. In the following example, the "AttributeNames" are contained in the list of image and annotation references ["source-ref", "class"]. The corresponding label value is "0" for the first image and "1" for the second image:

```json
{ "source-ref":"s3://image/filename1.jpg", "class":"0" }
{ "source-ref":"s3://image/filename2.jpg", "class":"1", "class-metadata": { "class-name": "cat", "type" : "groundtruth/image-classification"} }
```

The order of "AttributeNames" in the input files matters when training the ImageClassification algorithm. It accepts piped data in a specific order, with image first, followed by label. So the "AttributeNames" in this example are provided with "source-ref" first, followed by "class". When using the ImageClassification algorithm with Augmented Manifest, the value of the RecordWrapperType parameter must be "RecordIO".

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. Amazon SageMaker image classification models can be seeded only with another build-in image classification model trained in Amazon SageMaker.

To use a pretrained model, in the CreateTrainingJob (p. 931) request, specify the ChannelName as "model" in the InputDataConfig parameter. Set the ContentType for the model channel to application/x-sagemaker-model. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the num_layers, image_shape and num_classes input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.
For a sample notebook that shows how to use incremental training with the Amazon SageMaker image classification algorithm, see the End-to-End Incremental Training Image Classification Example. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 594).

Inference with the Image Format Algorithm

The generated models can be hosted for inference and support encoded .jpg and .png image formats as image/png, image/jpeg, and application/x-image content-type. The output is the probability values for all classes encoded in JSON format, or in JSON Lines text format for batch transform. The image classification model processes a single image per request and so outputs only one line in the JSON or JSON Lines format. The following is an example of a response in JSON Lines format:

```json
accept: application/jsonlines
\n{"prediction": [prob_0, prob_1, prob_2, prob_3, ...]}
```

For more details on training and inference, see the image classification sample notebook instances referenced in the introduction.

EC2 Instance Recommendation for the Image Classification Algorithm

For image classification, we support the following GPU instances for training: ml.p2.xlarge, ml.p2.8xlarge, ml.p2.16xlarge, ml.p3.2xlarge, ml.p3.8xlarge and ml.p3.16xlarge. We recommend using GPU instances with more memory for training with large batch sizes. However, both CPU (such as C4) and GPU (such as P2 and P3) instances can be used for the inference. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training.

Both P2 and P3 instances are supported in the image classification algorithm.

Image Classification Sample Notebooks

For a sample notebook that uses the Amazon SageMaker image classification algorithm to train a model on the caltech-256 dataset and then to deploy it to perform inferences, see the End-to-End Multiclass Image Classification Example. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The example image classification notebooks are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

How Image Classification Works

The image classification algorithm takes an image as input and classifies it into one of the output categories. Deep learning has revolutionized the image classification domain and has achieved great performance. Various deep learning networks such as ResNet [1], DenseNet, inception, and so on, have been developed to be highly accurate for image classification. At the same time, there have been efforts to collect labeled image data that are essential for training these networks. ImageNet[2] is one such large dataset that has more than 11 million images with about 11,000 categories. Once a network is trained with ImageNet data, it can then be used to generalize with other datasets as well, by simple re-adjustment or fine-tuning. In this transfer learning approach, a network is initialized with weights (in this example, trained on ImageNet), which can be later fine-tuned for an image classification task in a different dataset.

Image classification in Amazon SageMaker can be run in two modes: full training and transfer learning. In full training mode, the network is initialized with random weights and trained on user data from scratch. In transfer learning mode, the network is initialized with pre-trained weights and just the top fully connected layer is initialized with random weights. Then, the whole network is fine-tuned with new
data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data.

### Image Classification Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes</td>
<td>Number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset. <strong>Required</strong> Valid values: positive integer</td>
</tr>
<tr>
<td>num_training_samples</td>
<td>Number of training examples in the input dataset. If there is a mismatch between this value and the number of samples in the training set, then the behavior of the lr_scheduler_step parameter is undefined and distributed training accuracy might be affected. <strong>Required</strong> Valid values: positive integer</td>
</tr>
<tr>
<td>augmentation_type</td>
<td>Data augmentation type. The input images can be augmented in multiple ways as specified below. <strong>Optional</strong> Valid values: crop, crop_color, or crop_color_transform. Default value: no default value</td>
</tr>
<tr>
<td>beta_1</td>
<td>The beta1 for adam, that is the exponential decay rate for the first moment estimates. <strong>Optional</strong> Valid values: float. Range in [0, 1]. Default value: 0.9</td>
</tr>
<tr>
<td>beta_2</td>
<td>The beta2 for adam, that is the exponential decay rate for the second moment estimates. <strong>Optional</strong></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Valid values:</strong> float. Range in ([0, 1]).</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 0.999</td>
</tr>
<tr>
<td><strong>checkpoint_frequency</strong></td>
<td><strong>Period to store model parameters (in number of epochs).</strong> Note that all checkpoint files are saved as part of the final model file &quot;model.tar.gz&quot; and uploaded to S3 to the specified model location. This increases the size of the model file proportionally to the number of checkpoints saved during training. <strong>Optional</strong> <strong>Valid values:</strong> positive integer no greater than epochs. <strong>Default value:</strong> no default value (Save checkpoint at the epoch that has the best validation accuracy)</td>
</tr>
<tr>
<td><strong>early_stopping</strong></td>
<td><strong>True</strong> to use early stopping logic during training. <strong>False</strong> not to use it. <strong>Optional</strong> <strong>Valid values:</strong> True or False <strong>Default value:</strong> False</td>
</tr>
<tr>
<td></td>
<td><strong>early_stopping_min_epochs</strong> <strong>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when early_stopping = True.</strong> <strong>Optional</strong> <strong>Valid values:</strong> positive integer <strong>Default value:</strong> 10</td>
</tr>
<tr>
<td></td>
<td><strong>early_stopping_patience</strong> <strong>The number of epochs to wait before ending training if no improvement is made in the relevant metric. It is used only when early_stopping = True.</strong> <strong>Optional</strong> <strong>Valid values:</strong> positive integer <strong>Default value:</strong> 5</td>
</tr>
<tr>
<td></td>
<td><strong>early_stopping_tolerance</strong> <strong>Relative tolerance to measure an improvement in accuracy validation metric. If the ratio of the improvement in accuracy divided by the previous best accuracy is smaller than the early_stopping_tolerance value set, early stopping considers there is no improvement. It is used only when early_stopping = True.</strong> <strong>Optional</strong> <strong>Valid values:</strong> (0 \leq \text{float} \leq 1) <strong>Default value:</strong> 0.0</td>
</tr>
</tbody>
</table>

291
## Parameter Name | Description
--- | ---
**epochs** | Number of training epochs.  
  **Optional**  
  Valid values: positive integer  
  Default value: 30

**eps** | The epsilon for adam and rmsprop. It is usually set to a small value to avoid division by 0.  
  **Optional**  
  Valid values: float. Range in [0, 1].  
  Default value: 1e-8

**gamma** | The gamma for rmsprop, the decay factor for the moving average of the squared gradient.  
  **Optional**  
  Valid values: float. Range in [0, 1].  
  Default value: 0.9

**image_shape** | The input image dimensions, which is the same size as the input layer of the network. The format is defined as 'num_channels, height, width'. The image dimension can take on any value as the network can handle varied dimensions of the input. However, there may be memory constraints if a larger image dimension is used. Typical image dimensions for image classification are '3, 224, 224'. This is similar to the ImageNet dataset.  
  **Optional**  
  Valid values: string  
  Default value: '3, 224, 224'
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **kv_store**   | Weight update synchronization mode during distributed training. The weight updates can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See distributed training in MXNet for more details.  
  
  This parameter is not applicable to single machine training.  
  
  - **dist_sync**: The gradients are synchronized after every batch with all the workers. With dist_sync, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like local with batch size nb  
  - **dist_async**: Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed.  
  
  **Optional**  
  
  Valid values: dist_sync or dist_async  
  
  Default value: no default value |
| **learning_rate** | Initial learning rate.  
  
  **Optional**  
  
  Valid values: float. Range in [0, 1].  
  
  Default value: 0.1 |
| **lr_scheduler_factor** | The ratio to reduce learning rate used in conjunction with the lr_scheduler_step parameter, defined as lr_new = lr_old * lr_scheduler_factor.  
  
  **Optional**  
  
  Valid values: float. Range in [0, 1].  
  
  Default value: 0.1 |
| **lr_scheduler_step** | The epochs at which to reduce the learning rate. As explained in the lr_scheduler_factor parameter, the learning rate is reduced by lr_scheduler_factor at these epochs. For example, if the value is set to "10, 20", then the learning rate is reduced by lr_scheduler_factor after 10th epoch and again by lr_scheduler_factor after 20th epoch. The epochs are delimited by ",".  
  
  **Optional**  
  
  Valid values: string  
  
  Default value: no default value |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. In a single-machine multi-GPU setting, each GPU handles mini_batch_size/num_gpu training samples. For the multi-machine training in dist_sync mode, the actual batch size is mini_batch_size*number of machines. See MXNet docs for more details.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 32</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for sgd and nag, ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>multi_label</td>
<td>Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>num_layers</td>
<td>Number of layers for the network. For data with large image size (for example, 224x224 - like ImageNet), we suggest selecting the number of layers from the set [18, 34, 50, 101, 152, 200]. For data with small image size (for example, 28x28 - like CIFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200].</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110]</td>
</tr>
<tr>
<td></td>
<td>Default value: 152</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: One of <strong>sgd</strong>, <strong>adam</strong>, <strong>rmsprop</strong>, or <strong>nag</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>sgd</strong>: Stochastic gradient descent</td>
</tr>
<tr>
<td></td>
<td>• <strong>adam</strong>: Adaptive momentum estimation</td>
</tr>
<tr>
<td></td>
<td>• <strong>rmsprop</strong>: Root mean square propagation</td>
</tr>
<tr>
<td></td>
<td>• <strong>nag</strong>: Nesterov accelerated gradient</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>sgd</strong></td>
</tr>
<tr>
<td>precision_dtype</td>
<td>The precision of the weights used for training. The algorithm can use either single precision (<strong>float32</strong>) or half precision (<strong>float16</strong>) for the weights. Using half-precision for weights results in reduced memory consumption.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <strong>float32</strong> or <strong>float16</strong></td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>float32</strong></td>
</tr>
<tr>
<td>resize</td>
<td>Resizes the image before using it for training. The images are resized so that the shortest side has the number of pixels specified by this parameter. If the parameter is not set, then the training data is used without resizing.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>top_k</td>
<td>Reports the top-k accuracy during training. This parameter has to be greater than 1, since the top-1 training accuracy is the same as the regular training accuracy that has already been reported.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer larger than 1.</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
</tbody>
</table>
### Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| use_pretrained_model | Flag to use pre-trained model for training. If set to 1, then the pretrained model with the corresponding number of layers is loaded and used for training. Only the top FC layer are reinitialized with random weights. Otherwise, the network is trained from scratch. *Optional*  
Valid values: 0 or 1  
Default value: 0 |
| use_weighted_loss    | Flag to use weighted cross-entropy loss for multi-label classification (used only when `multi_label = 1`), where the weights are calculated based on the distribution of classes. *Optional*  
Valid values: 0 or 1  
Default value: 0 |
| weight_decay         | The coefficient weight decay for sgd and nag, ignored for other optimizers. *Optional*  
Valid values: float. Range in [0, 1].  
Default value: 0.0001 |

### Tune an Image Classification Model

*Automatic model tuning,* also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see *Perform Automatic Model Tuning* (p. 555).

### Metrics Computed by the Image Classification Algorithm

The image classification algorithm is a supervised algorithm. It reports an accuracy metric that is computed during training. When tuning the model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:accuracy</td>
<td>The ratio of the number of correct predictions to the total number of predictions made.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### Tunable Image Classification Hyperparameters

Tune an image classification model with the following hyperparameters. The hyperparameters that have the greatest impact on image classification objective metrics are: `mini_batch_size`, `learning_rate`,...
and optimizer. Tune the optimizer-related hyperparameters, such as momentum, weight_decay, beta_1, beta_2, eps, and gamma, based on the selected optimizer. For example, use beta_1 and beta_2 only when adam is the optimizer.

For more information about which hyperparameters are used in each optimizer, see Image Classification Hyperparameters (p. 290).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_1</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.999</td>
</tr>
<tr>
<td>beta_2</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.999</td>
</tr>
<tr>
<td>eps</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-8, MaxValue: 1.0</td>
</tr>
<tr>
<td>gamma</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-8, MaxValue: 0.999</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.5</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 512</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'rmsprop', 'nag']</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
</tbody>
</table>

**IP Insights Algorithm**

Amazon SageMaker IP Insights is an unsupervised learning algorithm that learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers. You can use it to identify a user attempting to log into a web service from an anomalous IP address, for example. Or you can use it to identify an account that is attempting to create computing resources from an unusual IP address. Trained IP Insight models can be hosted at an endpoint for making real-time predictions or used for processing batch transforms.

Amazon SageMaker IP insights ingests historical data as (entity, IPv4 Address) pairs and learns the IP usage patterns of each entity. When queried with an (entity, IPv4 Address) event, an Amazon SageMaker IP Insights model returns a score that infers how anomalous the pattern of the event is. For example, when a user attempts to log in from an IP address, if the IP Insights score is high enough, a web login server might decide to trigger a multi-factor authentication system. In more advanced solutions, you can feed the IP Insights score into another machine learning model. For example, you can combine the IP Insight score with other features to rank the findings of another security system, such as those from Amazon GuardDuty.

The Amazon SageMaker IP Insights algorithm can also learn vector representations of IP addresses, known as *embeddings*. You can use vector-encoded embeddings as features in downstream machine learning tasks that use the information observed in the IP addresses. For example, you can use them in tasks such as measuring similarities between IP addresses in clustering and visualization tasks.
Input/Output Interface for the IP Insights Algorithm

Training and Validation
The Amazon SageMaker IP Insights algorithm supports training and validation data channels. It uses the optional validation channel to compute an area-under-curve (AUC) score on a predefined negative sampling strategy. The AUC metric validates how well the model discriminates between positive and negative samples. Training and validation data content types need to be in text/csv format. The first column of the CSV data is an opaque string that provides a unique identifier for the entity. The second column is an IPv4 address in decimal-dot notation. IP Insights currently supports only File mode. For more information and some examples, see IP Insights Training Data Formats (p. 304).

Inference
For inference, IP Insights supports text/csv, application/json, and application/jsonlines data content types. For more information about the common data formats for inference provided by Amazon SageMaker, see Common Data Formats for Inference (p. 232). IP Insights inference returns output formatted as either application/json or application/jsonlines. Each record in the output data contains the corresponding dot_product (or compatibility score) for each input data point. For more information and some examples, see IP Insights Inference Data Formats (p. 305).

EC2 Instance Recommendation for the IP Insights Algorithm
The Amazon SageMaker IP Insights algorithm can run on both GPU and CPU instances. For training jobs, we recommend using GPU instances. However, for certain workloads with large training datasets, distributed CPU instances might reduce training costs. For inference, we recommend using CPU instances.

GPU Instances for the IP Insights Algorithm
IP Insights supports all available GPUs. If you need to speed up training, we recommend starting with a single GPU instance, such as ml.p3.2xlarge, and then moving to a multi-GPU environment, such as ml.p3.8xlarge and ml.p3.16xlarge. Multi-GPUs automatically divide the mini batches of training data across themselves. If you switch from a single GPU to multiple GPUs, the mini_batch_size is divided equally into the number of GPUs used. You may want to increase the value of the mini_batch_size to compensate for this.

CPU Instances for the IP Insights Algorithm
The type of CPU instance that we recommend depends largely on the instance’s available memory and the model size. The model size is determined by two hyperparameters: vector_dim and num_entity_vectors. The maximum supported model size is 8 GB. The following table lists typical EC2 instance types that you would deploy based on these input parameters for various model sizes. In Table 1, the value for vector_dim in the first column range from 32 to 2048 and the values for num_entity_vectors in the first row range from 10,000 to 50,000.
The values for the `mini_batch_size`, `num_ip_encoder_layers`, `random_negative_sampling_rate`, and `shuffled_negative_sampling_rate` hyperparameters also affect the amount of memory required. If these values are large, you might need to use a larger instance type than normal.

**IP Insights Sample Notebooks**

For a sample notebook that shows how to train the Amazon SageMaker IP Insights algorithm and perform inferences with it, see [An Introduction to the Amazon SageMaker IP Insights Algorithm](https://docs.aws.amazon.com). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances](https://docs.aws.amazon.com). After creating a notebook instance, choose the `SageMaker Examples` tab to see a list of all the Amazon SageMaker examples. To open a notebook, choose its `Use` tab and choose `Create copy`.

**How IP Insights Works**

Amazon SageMaker IP Insights is an unsupervised algorithm that consumes observed data in the form of (entity, IPv4 address) pairs that associates entities with IP addresses. IP Insights determines how likely it is that an entity would use a particular IP address by learning latent vector representations for both entities and IP addresses. The distance between these two representations can then serve as the proxy for how likely this association is.

The IP Insights algorithm uses a neural network to learn the latent vector representations for entities and IP addresses. Entities are first hashed to a large but fixed hash space and then encoded by a simple embedding layer. Character strings such as user names or account IDs can be fed directly into IP Insights as they appear in log files. You don't need to preprocess the data for entity identifiers. You can provide entities as an arbitrary string value during both training and inference. The hash size should be configured with a value that is high enough to insure that the number of collisions, which occur when distinct entities are mapped to the same latent vector, remain insignificant. For more information about how to select appropriate hash sizes, see [Feature Hashing for Large Scale Multitask Learning](https://docs.aws.amazon.com). For representing IP addresses, on the other hand, IP Insights uses a specially designed encoder network to uniquely represent each possible IPv4 address by exploiting the prefix structure of IP addresses.

During training, IP Insights automatically generates negative samples by randomly pairing entities and IP addresses. These negative samples represent data that is less likely to occur in reality. The model is trained to discriminate between positive samples that are observed in the training data and these generated negative samples. More specifically, the model is trained to minimize the cross entropy, also known as the log loss, defined as follows:
\[ L = \frac{1}{N} \sum_n [y_n \log p_n + (1 - y_n) \log (1 - p_n)] \]

\( y_n \) is the label that indicates whether the sample is from the real distribution governing observed data (\( y_n = 1 \)) or from the distribution generating negative samples (\( y_n = 0 \)). \( p_n \) is the probability that the sample is from the real distribution, as predicted by the model.

Generating negative samples is an important process that is used to achieve an accurate model of the observed data. If negative samples are extremely unlikely, for example, if all of the IP addresses in negative samples are 10.0.0.0, then the model trivially learns to distinguish negative samples and fails to accurately characterize the actual observed dataset. To keep negative samples more realistic, IP Insights generates negative samples both by randomly generating IP addresses and randomly picking IP addresses from training data. You can configure the type of negative sampling and the rates at which negative samples are generated with the random_negative_sampling_rate and shuffled_negative_sampling_rate hyperparameters.

Given an nth (entity, IP address pair), the IP Insights model outputs a score, \( S_n \), that indicates how compatible the entity is with the IP address. This score corresponds to the log odds ratio for a given (entity, IP address) of the pair coming from a real distribution as compared to coming from a negative distribution. It is defined as follows:

\[ S_n = \log \left( \frac{P_{\text{real}}(n)}{P_{\text{neg}}(n)} \right) \]

The score is essentially a measure of the similarity between the vector representations of the nth entity and IP address. It can be interpreted as how much more likely it would be to observe this event in reality than in a randomly generated dataset. During training, the algorithm uses this score to calculate an estimate of the probability of a sample coming from the real distribution, \( p_n \), to use in the cross entropy minimization, where:

\[ p_n = \frac{1}{1 + e^{-S_n}} \]

**IP Insights Hyperparameters**

In the `CreateTransformJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker IP Insights algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_entity_vectors</td>
<td>The number of entity vector representations (entity embedding vectors) to train. Each entity in the training set is randomly assigned to one of these vectors using a hash function. Because of hash collisions, it might be possible to have multiple entities assigned to the same vector. This would cause the same vector to represent multiple entities. This generally has a negligible effect on model performance, as long as the collision rate is not too severe. To keep the collision rate low, set this value as high as possible. However, the model size, and, therefore, the memory requirement, for both training and inference, scales linearly with this hyperparameter. We recommend that you set this value to twice the number of unique entity identifiers. Required Valid values: 1 ≤ positive integer ≤ 250,000,000</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>vector_dim</td>
<td>The size of embedding vectors to represent entities and IP addresses. The larger the value, the more information that can be encoded using these representations. In practice, model size scales linearly with this parameter and limits how large the dimension can be. In addition, using vector representations that are too large can cause the model to overfit, especially for small training datasets. Overfitting occurs when a model doesn't learn any pattern in the data but effectively memorizes the training data and, therefore, cannot generalize well and performs poorly during inference. The recommended value is 128.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $4 \leq \text{positive integer} \leq 4096$</td>
</tr>
<tr>
<td>batch_metrics_publish_interval</td>
<td>The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer $\geq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 1,000</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of passes over the training data. The optimal value depends on your data size and learning rate. Typical values range from 5 to 100.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer $\geq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate for the optimizer. IP Insights use a gradient-descent-based Adam optimizer. The learning rate effectively controls the step size to update model parameters at each iteration. Too large a learning rate can cause the model to diverge because the training is likely to overshoot a minima. On the other hand, too small a learning rate slows down convergence. Typical values range from $1e^{-4}$ to $1e^{-1}$.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1e^{-6} \leq \text{float} \leq 10.0$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The number of examples in each mini batch. The training procedure processes data in mini batches. The optimal value depends on the number of unique account identifiers in the dataset. In general, the larger the mini_batch_size, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large mini_batch_size, the training is more likely to converge to a poor local minimum and perform relatively worse for inference.</td>
</tr>
<tr>
<td>num_ip_encoder_layers</td>
<td>The number of fully connected layers used to encode the IP address embedding. The larger the number of layers, the greater the model's capacity to capture patterns among IP addresses. However, using a large number of layers increases the chance of overfitting.</td>
</tr>
<tr>
<td>random_negative_sampling_rate</td>
<td>The number of random negative samples, R, to generate per input example. The training procedure relies on negative samples to prevent the vector representations of the model collapsing to a single point. Random negative sampling generates R random IP addresses for each input account in the mini batch. The sum of the random_negative_sampling_rate (R) and shuffled_negative_sampling_rate (S) must be in the interval: 1 ≤ R + S ≤ 500.</td>
</tr>
</tbody>
</table>
### Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shuffled_negative_sampling_rate</td>
<td>The number of shuffled negative samples, S, to generate per input example. In some cases, it helps to use more realistic negative samples that are randomly picked from the training data itself. This kind of negative sampling is achieved by shuffling the data within a mini batch. Shuffled negative sampling generates S negative IP addresses by shuffling the IP address and account pairings within a mini batch. The sum of the random_negative_sampling_rate (R) and shuffled_negative_sampling_rate (S) must be in the interval: 1 ≤ R + S ≤ 500.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ positive integer ≤ 500</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0.0 ≤ float ≤ 10.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.00001</td>
</tr>
</tbody>
</table>

### Tune an IP Insights Model

**Automatic model tuning**, also called hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

### Metrics Computed by the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm is an unsupervised learning algorithm that learns associations between IP addresses and entities. The algorithm trains a discriminator model, which learns to separate observed data points (*positive samples*) from randomly generated data points (*negative samples*). Automatic model tuning on IP Insights helps you find the model that can most accurately distinguish between unlabeled validation data and automatically generated negative samples. The model accuracy on the validation dataset is measured by the area under the receiver operating characteristic (ROC) curve. This validation:discriminator_auc metric can take values between 0.0 and 1.0, where 1.0 indicates perfect accuracy.

The IP Insights algorithm computes a validation:discriminator_auc metric during validation, the value of which is used as the objective function to optimize for hyperparameter tuning.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:discriminator_auc</td>
<td>Area under the ROC curve on the validation dataset. The validation dataset is not labeled. AUC</td>
<td>Maximize</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Optimization Direction</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>is a metric that describes the model's ability to discriminate validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data points from randomly generated data points.</td>
<td></td>
</tr>
</tbody>
</table>

**Tunable IP Insights Hyperparameters**

You can tune the following hyperparameters for the Amazon SageMaker IP Insights algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 100</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-4, MaxValue: 0.1</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 100, MaxValue: 50000</td>
</tr>
<tr>
<td>num_entity_vectors</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 10000, MaxValue: 1000000</td>
</tr>
<tr>
<td>num_ip_encoder_layers</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 10</td>
</tr>
<tr>
<td>random_negative_sampling</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 0, MaxValue: 10</td>
</tr>
<tr>
<td>shuffled_negative_sampling</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 0, MaxValue: 10</td>
</tr>
<tr>
<td>vector_dim</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 256</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
</tbody>
</table>

**IP Insights Data Formats**

This section provides examples of the available input and output data formats used by the IP Insights algorithm during training and inference.

**Topics**

- IP Insights Training Data Formats (p. 304)
- IP Insights Inference Data Formats (p. 305)

**IP Insights Training Data Formats**

The following are the available data input formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input training format described in Common Data Formats for Training (p. 228). However, the Amazon SageMaker IP Insights algorithm currently supports only the CSV data input format.
IP Insights Training Data Input Formats

**INPUT: CSV**

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

IP Insights Inference Data Formats

The following are the available input and output formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats for Inference (p. 232). However, the Amazon SageMaker IP Insights algorithm does not currently support RecordIO format.

IP Insights Input Request Formats

**INPUT: CSV Format**

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

**INPUT: JSON Format**

JSON data can be provided in different formats. IP Insights follows the common Amazon SageMaker formats. For more information about inference formats, see Common Data Formats for Inference (p. 232).

content-type: application/json

```
{
  "instances": [
    {
      "data": {
        "features": {
          "values": ["entity_id_1", "192.168.1.2"]
        }
      }
    },
    {
      "features": ["entity_id_2", "10.10.1.2"]
    }
  ]
}
```

**INPUT: JSONLINES Format**

The JSON Lines content type is useful for running batch transform jobs. For more information on Amazon SageMaker inference formats, see Common Data Formats for Inference (p. 232). For more information on running batch transform jobs, see Get Inferences for an Entire Dataset with Batch Transform (p. 11).

content-type: application/jsonlines
IP Insights Output Response Formats

OUTPUT: JSON Response Format

The default output of the Amazon SageMaker IP Insights algorithm is the dot_product between the input entity and IP address. The dot_product signifies how compatible the model considers the entity and IP address. The dot_product is unbounded. To make predictions about whether an event is anomalous, you need to set a threshold based on your defined distribution. For information about how to use the dot_product for anomaly detection, see the An Introduction to the Amazon SageMakerIP Insights Algorithm.

accept: application/json

```json
{
  "predictions": [
    {
      "dot_product": 0.0,
      "entity_embedding": [1.0, 0.0, 0.0],
      "ip_embedding": [0.0, 1.0, 0.0]
    },
    {
      "dot_product": 2.0,
      "entity_embedding": [1.0, 0.0, 1.0],
      "ip_embedding": [1.0, 0.0, 1.0]
    }
  ]
}
```

Advanced users can access the model's learned entity and IP embeddings by providing the additional content-type parameter verbose=True to the Accept heading. You can use the entity_embedding and ip_embedding for debugging, visualizing, and understanding the model. Additionally, you can use these embeddings in other machine learning techniques, such as classification or clustering.

accept: application/json;verbose=True

```json
{
  "predictions": [
    {
      "dot_product": 0.0,
      "entity_embedding": [1.0, 0.0, 0.0],
      "ip_embedding": [0.0, 1.0, 0.0]
    },
    {
      "dot_product": 2.0,
      "entity_embedding": [1.0, 0.0, 1.0],
      "ip_embedding": [1.0, 0.0, 1.0]
    }
  ]
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonl

```json
{"dot_product": 0.0}
{"dot_product": 2.0}
```

accept: application/jsonlines; verbose=True

```json
{"dot_product": 0.0, "entity_embedding": [1.0, 0.0, 0.0], "ip_embedding": [0.0, 1.0, 0.0]}
{"dot_product": 2.0, "entity_embedding": [1.0, 0.0, 1.0], "ip_embedding": [1.0, 0.0, 1.0]}
```
K-Means Algorithm

K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

Amazon SageMaker uses a modified version of the web-scale k-means clustering algorithm. Compared with the original version of the algorithm, the version used by Amazon SageMaker is more accurate. Like the original algorithm, it scales to massive datasets and delivers improvements in training time. To do this, the version used by Amazon SageMaker streams mini-batches (small, random subsets) of the training data. For more information about mini-batch k-means, see Web-scale k-means Clustering.

The k-means algorithm expects tabular data, where rows represent the observations that you want to cluster, and the columns represent attributes of the observations. The \( n \) attributes in each row represent a point in \( n \)-dimensional space. The Euclidean distance between these points represents the similarity of the corresponding observations. The algorithm groups observations with similar attribute values (the points corresponding to these observations are closer together). For more information about how k-means works in Amazon SageMaker, see How K-Means Clustering Works (p. 308).

Topics
- Input/Output Interface for the K-Means Algorithm (p. 307)
- EC2 Instance Recommendation for the K-Means Algorithm (p. 307)
- K-Means Sample Notebooks (p. 307)
- How K-Means Clustering Works (p. 308)
- K-Means Hyperparameters (p. 310)
- Tune a K-Means Model (p. 313)
- K-Means Response Formats (p. 314)

Input/Output Interface for the K-Means Algorithm

For training, the k-means algorithm expects data to be provided in the \texttt{train} channel (recommended \texttt{S3DataDistributionType=ShardedByS3Key}), with an optional \texttt{test} channel (recommended \texttt{S3DataDistributionType=FullyReplicated}) to score the data on. Both recordIO-wrapped-protobuf and CSV formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference, text/csv, application/json, and application/x-recordio-protobuf are supported. k-means returns a \texttt{closest_cluster} label and the \texttt{distance_to_cluster} for each observation.

For more information on input and output file formats, see K-Means Response Formats (p. 314) for inference and the K-Means Sample Notebooks (p. 307). The k-means algorithm does not support multiple instance learning, in which the training set consists of labeled “bags”, each of which is a collection of unlabeled instances.

EC2 Instance Recommendation for the K-Means Algorithm

We recommend training k-means on CPU instances. You can train on GPU instances, but should limit GPU training to \texttt{p*.xlarge} instances because only one GPU per instance is used.

K-Means Sample Notebooks

For a sample notebook that uses the Amazon SageMaker K-means algorithm to segment the population of counties in the United States by attributes identified using principle component analysis, see Analyze US census data for population segmentation using Amazon SageMaker. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see
Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its Use tab and select Create copy.

How K-Means Clustering Works

K-means is an algorithm that trains a model that groups similar objects together. The k-means algorithm accomplishes this by mapping each observation in the input dataset to a point in the $n$-dimensional space (where $n$ is the number of attributes of the observation). For example, your dataset might contain observations of temperature and humidity in a particular location, which are mapped to points $(t, h)$ in 2-dimensional space.

Note
Clustering algorithms are unsupervised. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

In k-means clustering, each cluster has a center. During model training, the k-means algorithm uses the distance of the point that corresponds to each observation in the dataset to the cluster centers as the basis for clustering. You choose the number of clusters ($k$) to create.

For example, suppose that you want to create a model to recognize handwritten digits and you choose the MNIST dataset for training. The dataset provides thousands of images of handwritten digits (0 through 9). In this example, you might choose to create 10 clusters, one for each digit (0, 1, ..., 9). As part of model training, the k-means algorithm groups the input images into 10 clusters.

Each image in the MNIST dataset is a 28x28-pixel image, with a total of 784 pixels. Each image corresponds to a point in a 784-dimensional space, similar to a point in a 2-dimensional space $(x,y)$. To find a cluster to which a point belongs, the k-means algorithm finds the distance of that point from all of the cluster centers. It then chooses the cluster with the closest center as the cluster to which the image belongs.

Note
Amazon SageMaker uses a customized version of the algorithm where, instead of specifying that the algorithm create $k$ clusters, you might choose to improve model accuracy by specifying extra cluster centers ($K = k * x$). However, the algorithm ultimately reduces these to $k$ clusters.

In Amazon SageMaker, you specify the number of clusters when creating a training job. For more information, see CreateTrainingJob (p. 931). In the request body, you add the HyperParameters string map to specify the $k$ and extra_center_factor strings.

The following is a summary of how k-means works for model training in Amazon SageMaker:

1. It determines the initial $K$ cluster centers.

   Note
   In the following topics, $K$ clusters refer to $k * x$, where you specify $k$ and $x$ when creating a model training job.

2. It iterates over input training data and recalculates cluster centers.

3. It reduces resulting clusters to $k$ (if the data scientist specified the creation of $k*x$ clusters in the request).

The following sections also explain some of the parameters that a data scientist might specify to configure a model training job as part of the HyperParameters string map.

Topics
- Step 1: Determine the Initial Cluster Centers (p. 309)
- Step 2: Iterate over the Training Dataset and Calculate Cluster Centers (p. 309)
- Step 3: Reduce the Clusters from $K$ to $k$ (p. 310)
Step 1: Determine the Initial Cluster Centers

When using k-means in Amazon SageMaker, the initial cluster centers are chosen from the observations in a small, randomly sampled batch. Choose one of the following strategies to determine how these initial cluster centers are selected:

- The random approach—Randomly choose $K$ observations in your input dataset as cluster centers. For example, you might choose a cluster center that points to the 784-dimensional space that corresponds to any 10 images in the MNIST training dataset.

- The k-means++ approach, which works as follows:
  1. Start with one cluster and determine its center. You randomly select an observation from your training dataset and use the point corresponding to the observation as the cluster center. For example, in the MNIST dataset, randomly choose a handwritten digit image. Then choose the point in the 784-dimensional space that corresponds to the image as your cluster center. This is cluster center 1.
  2. Determine the center for cluster 2. From the remaining observations in the training dataset, pick an observation at random. Choose one that is different than the one you previously selected. This observation corresponds to a point that is far away from cluster center 1. Using the MNIST dataset as an example, you do the following:
     - For each of the remaining images, find the distance of the corresponding point from cluster center 1. Square the distance and assign a probability that is proportional to the square of the distance. That way, an image that is different from the one that you previously selected has a higher probability of getting selected as cluster center 2.
     - Choose one of the images randomly, based on probabilities assigned in the previous step. The point that corresponds to the image is cluster center 2.
  3. Repeat Step 2 to find cluster center 3. This time, find the distances of the remaining images from cluster center 2.
  4. Repeat the process until you have the $K$ cluster centers.

To train a model in Amazon SageMaker, you create a training job. In the request, you provide configuration information by specifying the following HyperParameters string maps:

- To specify the number of clusters to create, add the $k$ string.
- For greater accuracy, add the optional extra_center_factor string.
- To specify the strategy that you want to use to determine the initial cluster centers, add the init_method string and set its value to random or k-means++.

For more information, see CreateTrainingJob (p. 931). For an example, see Create and Run a Training Job (AWS SDK for Python (Boto 3)) (p. 32).

You now have an initial set of cluster centers.

Step 2: Iterate over the Training Dataset and Calculate Cluster Centers

The cluster centers that you created in the preceding step are mostly random, with some consideration for the training dataset. In this step, you use the training dataset to move these centers toward the true cluster centers. The algorithm iterates over the training dataset, and recalculates the $K$ cluster centers.

1. Read a mini-batch of observations (a small, randomly chosen subset of all records) from the training dataset and do the following.

   **Note**
   When creating a model training job, you specify the batch size in the mini_batch_size string in the HyperParameters string map.
a. Assign all of the observations in the mini-batch to one of the clusters with the closest cluster center.

b. Calculate the number of observations assigned to each cluster. Then, calculate the proportion of new points assigned per cluster.

For example, consider the following clusters:

Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.

Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.

Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

Calculate the proportion of new points assigned to each of clusters as follows:

\[ p_1 = \text{proportion of points assigned to } c_1 = \frac{25}{(100+25)} \]
\[ p_2 = \text{proportion of points assigned to } c_2 = \frac{40}{(150+40)} \]
\[ p_3 = \text{proportion of points assigned to } c_3 = \frac{5}{(450+5)} \]

c. Compute the center of the new points added to each cluster:

\[ d_1 = \text{center of the new points added to cluster } 1 \]
\[ d_2 = \text{center of the new points added to cluster } 2 \]
\[ d_3 = \text{center of the new points added to cluster } 3 \]

d. Compute the weighted average to find the updated cluster centers as follows:

\[
\text{Center of cluster } 1 = ((1 - p_1) \times \text{center of cluster } 1) + (p_1 \times d_1) \\
\text{Center of cluster } 2 = ((1 - p_2) \times \text{center of cluster } 2) + (p_2 \times d_2) \\
\text{Center of cluster } 3 = ((1 - p_3) \times \text{center of cluster } 3) + (p_3 \times d_3)
\]

2. Read the next mini-batch, and repeat Step 1 to recalculate the cluster centers.

3. For more information about mini-batch k-means, see Web-Scale k-means Clustering.

Step 3: Reduce the Clusters from \( K \) to \( k \)

If the algorithm created \( K \) clusters—\( K = k \times x \) where \( x \) is greater than 1—then it reduces the \( K \) clusters to \( k \) clusters. (For more information, see extra_center_factor in the preceding discussion.) It does this by applying Lloyd’s method with kmeans++ initialization to the \( K \) cluster centers. For more information about Lloyd’s method, see k-means clustering.

K-Means Hyperparameters

In the CreateTrainingJob request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the k-means training algorithm provided by Amazon SageMaker. For more information about how k-means clustering works, see How K-Means Clustering Works.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>The number of required clusters. <strong>Required</strong></td>
</tr>
<tr>
<td>epochs</td>
<td>The number of passes done over the training data. <strong>Optional</strong></td>
</tr>
<tr>
<td>eval_metrics</td>
<td>A JSON list of metric types used to report a score for the model. <strong>Optional</strong></td>
</tr>
<tr>
<td>extra_center_factor</td>
<td>The algorithm creates K centers = num_clusters * extra_center_factor as it runs and reduces the number of centers from K to k when finalizing the model. <strong>Optional</strong></td>
</tr>
<tr>
<td>half_life_time_size</td>
<td>Used to determine the weight given to an observation when computing a cluster mean. This weight decays exponentially as more points are observed. When a point is first observed, it is assigned a weight of 1 when computing the cluster mean. The decay constant for the exponential decay function is chosen so that after observing half_life_time_size points, its weight is 1/2. If set to 0, there is no decay. <strong>Optional</strong></td>
</tr>
</tbody>
</table>

- **Valid values**: Positive integer
- **Default value**: 1
- **Valid values**: Either ['"msd"'] or ['"ssd"'] or ['"msd", "ssd"]
- **Default value**: ['"msd"']
- **Valid values**: Either a positive integer or auto
- **Default value**: auto
- **Valid values**: Non-negative integer
- **Default value**: 0
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_method</td>
<td>Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An alternative k-means++ method chooses the first cluster center at random. Then it spreads out the position of the remaining initial clusters by weighting the selection of centers with a probability distribution that is proportional to the square of the distance of the remaining data points from existing centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either random or kmeans++.</td>
</tr>
<tr>
<td></td>
<td>Default value: random</td>
</tr>
<tr>
<td>local_lloyd_init_method</td>
<td>The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either random or kmeans++.</td>
</tr>
<tr>
<td></td>
<td>Default value: kmeans++</td>
</tr>
<tr>
<td>local_lloyd_max_iter</td>
<td>The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
<tr>
<td>local_lloyd_num_trials</td>
<td>The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either a positive integer or auto.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>local_lloyd_tol</td>
<td>The tolerance for change in loss for early stopping of Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
</tbody>
</table>
Use Built-in Algorithms

Parameter Name | Description
--- | ---
mini_batch_size | The number of observations per mini-batch for the data iterator.
  | Optional
  | Valid values: Positive integer
  | Default value: 5000

Tune a K-Means Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker k-means algorithm is an unsupervised algorithm that groups data into clusters whose members are as similar as possible. Because it is unsupervised, it doesn’t use a validation dataset that hyperparameters can optimize against. But it does take a test dataset and emits metrics that depend on the squared distance between the data points and the final cluster centroids at the end of each training run. To find the model that reports the tightest clusters on the test dataset, you can use a hyperparameter tuning job. The clusters optimize the similarity of their members.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

Metrics Computed by the K-Means Algorithm

The k-means algorithm computes the following metrics during training. When tuning a model, choose one of these metrics as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:msd</td>
<td>Mean squared distances between each record in the test set and the closest center of the model.</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:ssd</td>
<td>Sum of the squared distances between each record in the test set and the closest center of the model.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Tunable K-Means Hyperparameters

Tune the Amazon SageMaker k-means model with the following hyperparameters. The hyperparameters that have the greatest impact on k-means objective metrics are: mini_batch_size, extra_center_factor, and init_method. Tuning the hyperparameter epochs generally results in minor improvements.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>epochs</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue:10</td>
</tr>
<tr>
<td>extra_center_factor</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 4, MaxValue:10</td>
</tr>
</tbody>
</table>
K-Means Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker k-means algorithm.

**JSON Response Format**

```json
{
    "predictions": [
        {
            "closest_cluster": 1.0,
            "distance_to_cluster": 3.0,
        },
        {
            "closest_cluster": 2.0,
            "distance_to_cluster": 5.0,
        },
        ....
    ]
}
```

**JSONLINES Response Format**

```json
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```

**RECORDIO Response Format**

```json
[
    Record = {
        features = {},
        label = {
            'closest_cluster': {
                keys: [],
                values: [1.0, 2.0]  # float32
            },
            'distance_to_cluster': {
                keys: [],
                values: [3.0, 5.0]  # float32
            }
        }
    }
]
```

K-Nearest Neighbors (k-NN) Algorithm

Amazon SageMaker k-nearest neighbors (k-NN) algorithm is an index-based algorithm. It uses a non-parametric method for classification or regression. For classification problems, the algorithm queries the k points that are closest to the sample point and returns the most frequently used label of their class as
the predicted label. For regression problems, the algorithm queries the \( k \) closest points to the sample point and returns the average of their feature values as the predicted value.

Training with the k-NN algorithm has three steps: sampling, dimension reduction, and index building. Sampling reduces the size of the initial dataset so that it fits into memory. For dimension reduction, the algorithm decreases the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency. We provide two methods of dimension reduction methods: random projection and the fast Johnson-Lindenstrauss transform. Typically, you use dimension reduction for high-dimensional (\( d > 1000 \)) datasets to avoid the “curse of dimensionality” that troubles the statistical analysis of data that becomes sparse as dimensionality increases. The main objective of k-NN's training is to construct the index. The index enables efficient lookups of distances between points whose values or class labels have not yet been determined and the \( k \) nearest points to use for inference.

**Topics**
- Input/Output Interface for the k-NN Algorithm (p. 315)
- k-NN Sample Notebooks (p. 316)
- How the k-NN Algorithm Works (p. 316)
- EC2 Instance Recommendation for the k-NN Algorithm (p. 317)
- k-NN Hyperparameters (p. 317)
- Tune a k-NN Model (p. 319)
- Data Formats for k-NN Training Input (p. 320)
- k-NN Request and Response Formats (p. 320)

**Input/Output Interface for the k-NN Algorithm**

Amazon SageMaker k-NN supports train and test data channels.

- Use a *train channel* for data that you want to sample and construct into the k-NN index.
- Use a *test channel* to emit scores in log files. Scores are listed as one line per mini-batch: accuracy for classifier, mean-squared error (mse) for regressor for score.

For training inputs, k-NN supports text/csv and application/x-recordio-protobuf data formats. For input type text/csv, the first label\_size columns are interpreted as the label vector for that row. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference inputs, k-NN supports the application/json, application/x-recordio-protobuf, and text/csv data formats. The text/csv format accepts a label\_size and encoding parameter. It assumes a label\_size of 0 and a UTF-8 encoding.

For inference outputs, k-NN supports the application/json and application/x-recordio-protobuf data formats. These two data formats also support a verbose output mode. In verbose output mode, the API provides the search results with the distances vector sorted from smallest to largest, and corresponding elements in the labels vector.

For batch transform, k-NN supports the application/jsonlines data format for both input and output. An example input is as follows:

```
content-type: application/jsonlines

{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

An example output is as follows:
For more information on input and output file formats, see Data Formats for k-NN Training 
Input (p. 320) for training, k-NN Request and Response Formats (p. 320) for inference, and the k-NN 
Sample Notebooks (p. 316).

k-NN Sample Notebooks

For a sample notebook that uses the Amazon SageMaker k-nearest neighbor algorithm to predict 
wilderness cover types from geological and forest service data, see the K-Nearest Neighbor Covertype .

Use a Jupyter notebook instance to run the example in Amazon SageMaker. To learn how to create 
and open a Jupyter notebook instance in Amazon SageMaker, see Use Amazon SageMaker Notebook 
Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker 
Examples tab to see a list of all the Amazon SageMaker example notebooks. Find K-Nearest Neighbor 
notebooks in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab 
and select Create copy.

How the k-NN Algorithm Works

Step 1: Sample

To specify the total number of data points to be sampled from the training dataset, use the 
\texttt{sample\_size} parameter. For example, if the initial dataset has 1,000 data points and the \texttt{sample\_size} 
is set to 100, where the total number of instances is 2, each worker would sample 50 points. A total set 
of 100 data points would be collected. Sampling runs in linear time with respect to the number of data 
points.

Step 2: Perform Dimension Reduction

The current implementation of k-NN has two methods of dimension reduction. You specify the method 
in the \texttt{dimension\_reduction\_type} hyperparameter. The \texttt{sign} method specifies a random projection, 
which uses a linear projection using a matrix of random signs, and the \texttt{fjlt} method specifies a fast 
Johnson-Lindenstrauss transform, a method based on the Fourier transform. Both methods preserve the 
L2 and inner product distances. The \texttt{fjlt} method should be used when the target dimension is large 
and has better performance with CPU inference. The methods differ in their computational complexity. 
The \texttt{sign} method requires \(O(ndk)\) time to reduce the dimension of a batch of \(n\) points of dimension \(d\) 
into a target dimension \(k\). The \texttt{fjlt} method requires \(O(nd \log(d))\) time, but the constants involved are 
larger. Using dimension reduction introduces noise into the data and this noise can reduce prediction 
accuracy.

Step 3: Build an Index

During inference, the algorithm queries the index for the k-nearest-neighbors of a sample point. Based 
on the references to the points, the algorithm makes the classification or regression prediction. It makes 
the prediction based on the class labels or values provided. k-NN provides three different types of 
indexes: a flat index, an inverted index, and an inverted index with product quantization. You specify the 
type with the \texttt{index\_type} parameter.

Serialize the Model

When the k-NN algorithm finishes training, it serializes three files to prepare for inference.

- \texttt{model\_algo-1}: Contains the serialized index for computing the nearest neighbors.
• `model_algo-1.labels`: Contains serialized labels (np.float32 binary format) for computing the predicted label based on the query result from the index.

• `model_algo-1.json`: Contains the JSON-formatted model metadata which stores the k and `predictor_type` hyper-parameters from training for inference along with other relevant state.

With the current implementation of k-NN, you can modify the metadata file to change the way predictions are computed. For example, you can change k to 10 or change `predictor_type` to `regressor`.

```
{
  "k": 5,
  "predictor_type": "classifier",
  "dimension_reduction": {
    "type": "sign",
    "seed": 3,
    "target_dim": 10,
    "input_dim": 20
  },
  "normalize": False,
  "version": "1.0"
}
```

### EC2 Instance Recommendation for the k-NN Algorithm

#### Instance Recommendation for Training with the k-NN Algorithm

To start, try running training on a CPU, using, for example, an ml.m5.2xlarge instance, or on a GPU using, for example, an ml.p2.xlarge instance.

#### Instance Recommendation for Inference with the k-NN Algorithm

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.

### k-NN Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>feature_dim</code></td>
<td>The number of features in the input data.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer.</td>
</tr>
<tr>
<td><code>k</code></td>
<td>The number of nearest neighbors.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>predictor_type</code></td>
<td>The type of inference to use on the data labels.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>classifier</code> for classification or <code>regressor</code> for regression.</td>
</tr>
<tr>
<td><code>sample_size</code></td>
<td>The number of data points to be sampled from the training data set.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>dimension_reduction_target</code></td>
<td>The target dimension to reduce to.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>dimension_reduction_type</strong></td>
<td>The type of dimension reduction method.</td>
</tr>
<tr>
<td><strong>faiss_index_ivf_nlists</strong></td>
<td>The number of centroids to construct in the index when index_type is faiss.IVFFlat or faiss.IVFPQ.</td>
</tr>
<tr>
<td><strong>faiss_index_pq_m</strong></td>
<td>The number of vector sub-components to construct in the index when index_type is set to faiss.IVFPQ.</td>
</tr>
<tr>
<td><strong>index_metric</strong></td>
<td>The metric to measure the distance between points when finding nearest neighbors. When training with index_type set to faiss.IVFPQ, the INNER_PRODUCT distance and COSINE similarity are not supported.</td>
</tr>
<tr>
<td><strong>index_type</strong></td>
<td>The type of index.</td>
</tr>
</tbody>
</table>
Tune a k-NN Model

The Amazon SageMaker k-nearest neighbors algorithm is a supervised algorithm. The algorithm consumes a test data set and emits a metric about the accuracy for a classification task or about the mean squared error for a regression task. These accuracy metrics compare the model predictions for their respective task to the ground truth provided by the empirical test data. To find the best model that reports the highest accuracy or lowest error on the test dataset, run a hyperparameter tuning job for k-NN.

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric appropriate for the prediction task of the algorithm. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric. The hyperparameters are used only to help estimate model parameters and are not used by the trained model to make predictions.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

Metrics Computed by the k-NN Algorithm

The k-nearest neighbors algorithm computes one of two metrics in the following table during training depending on the type of task specified by the predictor_type hyper-parameter.

- `classifier` specifies a classification task and computes `test:accuracy`
- `regressor` specifies a regression task and computes `test:mse`.

Choose the `predictor_type` value appropriate for the type of task undertaken to calculate the relevant objective metric when tuning a model.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:accuracy</td>
<td>When <code>predictor_type</code> is set to <code>classifier</code>, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The accuracy reported ranges from 0.0 (0%) to 1.0 (100%).</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:mse</td>
<td>When <code>predictor_type</code> is set to <code>regressor</code>, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The mean squared error is computed by comparing the two labels.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
Tunable k-NN Hyperparameters

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 1024</td>
</tr>
<tr>
<td>sample_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 256, MaxValue: 20000000</td>
</tr>
</tbody>
</table>

Data Formats for k-NN Training Input

All Amazon SageMaker built-in algorithms adhere to the common input training formats described in Common Data Formats - Training. This topic contains a list of the available input formats for the Amazon SageMaker k-nearest-neighbor algorithm.

CSV Data Format

content-type: text/csv; label_size=1

4,1.2,1.3,9.6,20.3

The first label_size columns are interpreted as the label vector for that row.

RECORDIO Data Format

content-type: application/x-recordio-protobuf

```json
[
  Record = {
    features = {
      'values': {
        values: [1.2, 1.3, 9.6, 20.3]  # float32
      }
    },
    label = {
      'values': {
        values: [4]  # float32
      }
    }
  }
]
```

k-NN Request and Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker k-nearest-neighbor algorithm.

INPUT: CSV Request Format

content-type: text/csv

1.2,1.3,9.6,20.3
This accepts a `label_size` or encoding parameter. It assumes a `label_size` of 0 and a utf-8 encoding.

**INPUT: JSON Request Format**
```
{    "instances": [        {"data": {"features": {"values": [-3, -1, -4, 2]}}},        {"features": [3.0, 0.1, 0.04, 0.002]}]
}
```

**INPUT: JSONLINES Request Format**
```
{"features": [1.5, 16.0, 14.0, 23.0]}  
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

**INPUT: RECORDIO Request Format**
```
[    Record = {        features = {            'values': {                values: [-3, -1, -4, 2]  # float32            }        },        label = {}    },    Record = {        features = {            'values': {                values: [3.0, 0.1, 0.04, 0.002]  # float32            }        },        label = {}    }
]
```

**OUTPUT: JSON Response Format**
```
{    "predictions": [        {"predicted_label": 0.0},        {"predicted_label": 2.0}    ]
}
```

**OUTPUT: JSONLINES Response Format**
```
{"predicted_label": 0.0}  
{"predicted_label": 2.0}
```
OUTPUT: VERBOSE JSON Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/json; verbose=true

```
{
  "predictions": [
    {
      "predicted_label": 0.0,
      "distances": [3.11792408, 3.89746071, 6.32548437],
      "labels": [0.0, 1.0, 0.0]
    },
    {
      "predicted_label": 2.0,
      "distances": [1.08470316, 3.04917915, 5.25393973],
      "labels": [2.0, 2.0, 0.0]
    }
  ]
}
```

OUTPUT: RECORDIO-PROTobuf Response Format

content-type: application/x-recordio-protobuf

```
[
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [0.0]  # float32
      }
    }
  },
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [2.0]  # float32
      }
    }
  }
]
```

OUTPUT: VERBOSE RECORDIO-PROTobuf Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/x-recordio-protobuf; verbose=true

```
[
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [0.0]  # float32
      },
      'distances': {
        values: [3.11792408, 3.89746071, 6.32548437]  # float32
      },
      'labels': {
```
values: [0.0, 1.0, 0.0]  # float32
}

},
Record ={
  features = {},
  label ={
    'predicted_label':{
      values: [0.0]  # float32
    },
    'distances':{
      values: [1.08470316, 3.04917915, 5.25393973]  # float32
    },
    'labels':{
      values: [2.0, 2.0, 0.0]  # float32
    }
  }
}
]

SAMPLE OUTPUT for the k-NN Algorithm

For regressor tasks:

[06/08/2018 20:15:33 INFO 140026520049408] #test_score (algo-1) : ('mse', 0.013333333333333334)

For classifier tasks:

[06/08/2018 20:15:46 INFO 140285487171328] #test_score (algo-1) : ('accuracy', 0.98666666666666669)

Latent Dirichlet Allocation (LDA) Algorithm

The Amazon SageMaker Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

The exact content of two documents with similar topic mixtures will not be the same. But overall, you would expect these documents to more frequently use a shared subset of words, than when compared with a document from a different topic mixture. This allows LDA to discover these word groups and use them to form topics. As an extremely simple example, given a set of documents where the only words that occur within them are: eat, sleep, play, meow, and bark, LDA might produce topics like the following:

<table>
<thead>
<tr>
<th>Topic</th>
<th>eat</th>
<th>sleep</th>
<th>play</th>
<th>meow</th>
<th>bark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Topic 2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

You can infer that documents that are more likely to fall into Topic 1 are about cats (who are more likely to meow and sleep), and documents that fall into Topic 2 are about dogs (who prefer to play and bark). These topics can be found even though the words dog and cat never appear in any of the texts.
Input/Output Interface for the LDA Algorithm

LDA expects data to be provided on the train channel, and optionally supports a test channel, which is scored by the final model. LDA supports both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats. For CSV, the data must be dense and have dimension equal to number of records * vocabulary size. LDA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

For inference, text/csv, application/json, and application/x-recordio-protobuf content types are supported. Sparse data can also be passed for application/json and application/x-recordio-protobuf. LDA inference returns application/json or application/x-recordio-protobuf predictions, which include the topic_mixture vector for each observation.

Please see the LDA Sample LDA Notebooks (p. 324) for more detail on training and inference formats.

EC2 Instance Recommendation for the LDA Algorithm

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

LDA Sample LDA Notebooks

For a sample notebook that shows how to train the Amazon SageMaker Latent Dirichlet Allocation algorithm on a dataset and then how to deploy the trained model to perform inferences about the topic mixtures in input documents, see the An Introduction to SageMaker LDA. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

How LDA Works

Amazon SageMaker LDA is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of different categories. These categories are themselves a probability distribution over the features. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. This is opposed to discriminative models, which attempt to learn how inputs map to outputs.

You can use LDA for a variety of tasks, from clustering customers based on product purchases to automatic harmonic analysis in music. However, it is most commonly associated with topic modeling in text corpuses. Observations are referred to as documents. The feature set is referred to as vocabulary. A feature is referred to as a word. And the resulting categories are referred to as topics.

Note

Lemmatization significantly increases algorithm performance and accuracy. Consider pre-processing any input text data.

An LDA model is defined by two parameters:
LDA is a "bag-of-words" model, which means that the order of words does not matter. LDA is a generative model where each document is generated word-by-word by choosing a topic mixture $\theta \sim \text{Dirichlet}(\alpha)$.

For each word in the document:

- Choose a topic $z \sim \text{Multinomial}(\theta)$
- Choose the corresponding topic-word distribution $\beta_z$.
- Draw a word $w \sim \text{Multinomial}(\beta_z)$.

When training the model, the goal is to find parameters $\alpha$ and $\beta$, which maximize the probability that the text corpus is generated by the model.

The most popular methods for estimating the LDA model use Gibbs sampling or Expectation Maximization (EM) techniques. The Amazon SageMaker LDA uses tensor spectral decomposition. This provides several advantages:

- **Theoretical guarantees on results.** The standard EM-method is guaranteed to converge only to local optima, which are often of poor quality.
- **Embarrassingly parallelizable.** The work can be trivially divided over input documents in both training and inference. The EM-method and Gibbs Sampling approaches can be parallelized, but not as easily.
- **Fast.** Although the EM-method has low iteration cost it is prone to slow convergence rates. Gibbs Sampling is also subject to slow convergence rates and also requires a large number of samples.

At a high-level, the tensor decomposition algorithm follows this process:

1. The goal is to calculate the spectral decomposition of a $V \times V \times V$ tensor, which summarizes the moments of the documents in our corpus. $V$ is vocabulary size (in other words, the number of distinct words in all of the documents). The spectral components of this tensor are the LDA parameters $\alpha$ and $\beta$, which maximize the overall likelihood of the document corpus. However, because vocabulary size tends to be large, this $V \times V \times V$ tensor is prohibitively large to store in memory.
2. Instead, it uses a $V \times V$ moment matrix, which is the two-dimensional analog of the tensor from step 1, to find a whitening matrix of dimension $V \times k$. This matrix can be used to convert the $V \times V$ moment matrix into a $k \times k$ identity matrix. $k$ is the number of topics in the model.
3. This same whitening matrix can then be used to find a smaller $k \times k \times k$ tensor. When spectrally decomposed, this tensor has components that have a simple relationship with the components of the $V \times V \times V$ tensor.
4. **Alternating Least Squares** is used to decompose the smaller $k \times k \times k$ tensor. This provides a substantial improvement in memory consumption and speed. The parameters $\alpha$ and $\beta$ can be found by "unwhitening" these outputs in the spectral decomposition.

After the LDA model’s parameters have been found, you can find the topic mixtures for each document. You use stochastic gradient descent to maximize the likelihood function of observing a given topic mixture corresponding to these data.

Topic quality can be improved by increasing the number of topics to look for in training and then filtering out poor quality ones. This is in fact done automatically in Amazon SageMaker LDA: 25% more topics are computed and only the ones with largest associated Dirichlet priors are returned. To perform
For more information about algorithms for LDA and the Amazon SageMaker implementation, see the following:


**LDA Hyperparameters**

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the LDA training algorithm provided by Amazon SageMaker. For more information, see How LDA Works (p. 324).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_topics</code></td>
<td>The number of topics for LDA to find within the data. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>feature_dim</code></td>
<td>The size of the vocabulary of the input document corpus. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>mini_batch_size</code></td>
<td>The total number of documents in the input document corpus. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>alpha0</code></td>
<td>Initial guess for the concentration parameter: the sum of the elements of the Dirichlet prior. Small values are more likely to generate sparse topic mixtures and large values (greater than 1.0) produce more uniform mixtures. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
---|---
max_restarts | The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm. Can be used to find better quality local minima at the expense of additional computation, but typically should not be adjusted. 
*Optional*
Valid values: Positive integer
Default value: 10

max_iterations | The maximum number of iterations to perform during the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.
*Optional*
Valid values: Positive integer
Default value: 1000

tol | Target error tolerance for the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.
*Optional*
Valid values: Positive float
Default value: 1e-8

---

### Tune an LDA Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

LDA is an unsupervised topic modeling algorithm that attempts to describe a set of observations (documents) as a mixture of different categories (topics). The “per-word log-likelihood” (PWLL) metric measures the likelihood that a learned set of topics (an LDA model) accurately describes a test document dataset. Larger values of PWLL indicate that the test data is more likely to be described by the LDA model.

For more information about model tuning, see [Perform Automatic Model Tuning](#).

### Metrics Computed by the LDA Algorithm

The LDA algorithm reports on a single metric during training: test:pwall. When tuning a model, choose this metric as the objective metric.
<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:pwll</td>
<td>Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

**Tunable LDA Hyperparameters**

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, alpha0 and num_topics, can affect the LDA objective metric (test:pwll). If you don’t already know the optimal values for these hyperparameters, which maximize per-word log-likelihood and produce an accurate LDA model, automatic model tuning can help find them.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha0</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 10</td>
</tr>
<tr>
<td>num_topics</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 150</td>
</tr>
</tbody>
</table>

**Linear Learner Algorithm**

*Linear models* are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples \((x, y)\). \(x\) is a high-dimensional vector and \(y\) is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to \(num\_classes - 1\). For regression problems, \(y\) is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector \(x\) to an approximation of the label \(y\).

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the Amazon SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from a validation set. You can also explore a large number of models and choose the best. The best model optimizes either of the following:

- Continuous objectives, such as mean square error, cross entropy loss, absolute error.
- Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.

Compared with methods that provide a solution for only continuous objectives, the Amazon SageMaker linear learner algorithm provides a significant increase in speed over naive hyperparameter optimization techniques. It is also more convenient.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification or regression) as arguments. The feature dimension is also required. For more information, see CreateTrainingJob (p. 931). You can specify additional parameters in the HyperParameters string map of the request body. These parameters control the optimization procedure, or specifics of the objective function that you train on. For example, the number of epochs, regularization, and loss type.

**Topics**

- Input/Output Interface for the Linear Learner Algorithm (p. 329)
Input/Output Interface for the Linear Learner Algorithm

The Amazon SageMaker linear learner algorithm supports three data channels: train, validation (optional), and test (optional). If you provide validation data, it should be FullyReplicated. The algorithm logs validation loss at every epoch, and uses a sample of the validation data to calibrate and select the best model. If you don't provide validation data, the algorithm uses a sample of the training data to calibrate and select the model. If you provide test data, the algorithm logs include the test score for the final model.

For training, the linear learner algorithm supports both recordIO-wrapped protobuf and CSV formats. For the application/x-recordio-protobuf input type, only Float32 tensors are supported. For the text/csv input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference, the linear learner algorithm supports the application/json, application/x-recordio-protobuf, and text/csv formats. When you make predictions on new data, the format of the response depends on the type of model. For regression (predictor_type='regressor'), the score is the prediction produced by the model. For classification (predictor_type='binary_classifier' or predictor_type='multiclass_classifier'), the model returns a score and also a predicted_label. The predicted_label is the class predicted by the model and the score measures the strength of that prediction.

- For binary classification, predicted_label is 0 or 1, and score is a single floating point number correspond class.
- For multiclass classification, the predicted_class will be an integer from 0 to num_classes-1, and the score will be a list of one floating point number per class.

To interpret the score in classification problems, you have to consider the loss function used. If the loss hyperparameter value is logistic for binary classification or softmax_loss for multiclass classification, then the score can be interpreted as the probability of the corresponding class. These are the loss values used by the linear learner when the loss value is auto default value. But if the loss is set to hinge_loss, then the score cannot be interpreted as a probability. This is because hinge loss corresponds to a Support Vector Classifier, which does not produce probability estimates.

For more information on input and output file formats, see Linear Learner Response Formats (p. 341). For more information on inference formats, and the Linear Learner Sample Notebooks (p. 329).

EC2 Instance Recommendation for the Linear Learner Algorithm

You can train the linear learner algorithm on single- or multi-machine CPU and GPU instances. During testing, we have not found substantial evidence that multi-GPU computers are faster than single-GPU computers. Results can vary, depending on your specific use case.

Linear Learner Sample Notebooks

For a sample notebook that uses the Amazon SageMaker linear learner algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Linear Learner with
MNIST. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After you have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all of the Amazon SageMaker samples. The topic modeling example notebooks using the linear learning algorithm are located in the Introduction to Amazon algorithms section. To open a notebook, choose its Use tab and choose Create copy.

How Linear Learner Works

There are three steps involved in the implementation of the linear learner algorithm: preprocess, train, and validate.

Step 1: Preprocess

Normalization, or feature scaling, is an important preprocessing step for certain loss functions that ensures the model being trained on a dataset does not become dominated by the weight of a single feature. The Amazon SageMaker Linear Learner algorithm has a normalization option to assist with this preprocessing step. If normalization is turned on, the algorithm first goes over a small sample of the data to learn the mean value and standard deviation for each feature and for the label. Each of the features in the full dataset is then shifted to have mean of zero and scaled to have a unit standard deviation.

Note

For best results, ensure your data is shuffled before training. Training with unshuffled data may cause training to fail.

You can configure whether the linear learner algorithm normalizes the feature data and the labels using the normalize_data and normalize_label hyperparameters respectively. Normalization is enabled by default for both features and labels for regression. Only the features can be normalized for binary classification and this is the default behavior.

Step 2: Train

With the linear learner algorithm, you train with a distributed implementation of stochastic gradient descent (SGD). You can control the optimization process by choosing the optimization algorithm. For example, you can choose to use Adam, AdaGrad, stochastic gradient descent, or other optimization algorithms. You also specify their hyperparameters, such as momentum, learning rate, and the learning rate schedule. If you aren't sure which algorithm or hyperparameter value to use, choose a default that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives. For example, you vary L1 or L2 regularization and try out different optimizer settings.

Step 3: Validate and Set the Threshold

When training multiple models in parallel, the models are evaluated against a validation set to select the most optimal model once training is complete. For regression, the most optimal model is the one that achieves the best loss on the validation set. For classification, a sample of the validation set is used to calibrate the classification threshold. The most optimal model selected is the one that achieves the best binary classification selection criteria on the validation set. Examples of such criteria include the F1 measure, accuracy, and cross-entropy loss.

Note

If the algorithm is not provided a validation set, then evaluating and selecting the most optimal model is not possible. To take advantage of parallel training and model selection ensure you provide a validation set to the algorithm.

Linear Learner Hyperparameters

The following table contains the hyperparameters for the linear learner algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required
hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The number of features in the input data.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td>num_classes</td>
<td>The number of classes for the response variable. The algorithm assumes that classes are labeled 0, ..., num_classes - 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong> when predictor_type is multiclass_classifier. Otherwise, the algorithm ignores it.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Integers from 3 to 1,000,000</td>
</tr>
<tr>
<td>predictor_type</td>
<td>Specifies the type of target variable as a binary classification, multiclass classification, or regression.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: binary_classifier, multiclass_classifier, or regressor</td>
</tr>
<tr>
<td>accuracy_top_k</td>
<td>When computing the top-k accuracy metric for multiclass classification, the value of k. If the model assigns one of the top-k scores to the true label, an example is scored as correct.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integers</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>balance_multiclass_weights</td>
<td>Specifies whether to use class weights, which give each class equal importance in the loss function. Used only when the predictor_type is multiclass_classifier.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: true, false</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>beta_1</td>
<td>The exponential decay rate for first-moment estimates. Applies only when the optimizer value is adam.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or floating-point value between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>beta_2</td>
<td>The exponential decay rate for second-moment estimates. Applies only when the optimizer value is adam.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
</tbody>
</table>

331
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias_lr_mult</td>
<td>Allows a different learning rate for the bias term. The actual learning rate for the bias is learning_rate * bias_lr_mult. <strong>Optional</strong></td>
</tr>
<tr>
<td>bias_wd_mult</td>
<td>Allows different regularization for the bias term. The actual L2 regularization weight for the bias is wd * bias_wd_mult. By default, there is no regularization on the bias term. <strong>Optional</strong></td>
</tr>
</tbody>
</table>
| binary_classifier_model_selection_criteria | When predictor_type is set to binary_classifier, the model evaluation criteria for the validation dataset (or for the training dataset if you don't provide a validation dataset). Criteria include:  
- accuracy—The model with the highest accuracy.  
- f_beta—The model with the highest F1 score. The default is F1.  
- precision_at_target_recall—The model with the highest precision at a given recall target.  
- recall_at_target_precision—The model with the highest recall at a given precision target.  
- loss_function—The model with the lowest value of the loss function used in training. **Optional**  
<p>| Valid values: accuracy, f_beta, precision_at_target_recall, recall_at_target_precision, or loss_function | Default value: accuracy |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>early_stopping_patience</td>
<td>The number of epochs to wait before ending training. If you have provided a value for binary_classifier_model_selection_criteria, the metric is that value. Otherwise, the metric is the same as the value specified for the loss hyperparameter. The metric is evaluated on the validation data. If you haven't provided validation data, the metric is always the same as the value specified for the loss hyperparameter and is evaluated on the training data. To disable early stopping, set early_stopping_patience to a value greater than the value specified for epochs. <strong>Optional</strong> Valid values: Positive integer Default value: 3</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>The relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping considers the improvement to be zero. <strong>Optional</strong> Valid values: Positive floating-point integer Default value: 0.001</td>
</tr>
<tr>
<td>epochs</td>
<td>The maximum number of passes over the training data. <strong>Optional</strong> Valid values: Positive integer Default value: 15</td>
</tr>
<tr>
<td>f_beta</td>
<td>The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if the value specified for binary_classifier_model_selection_criteria is f_beta. <strong>Optional</strong> Valid values: Positive floating-point integers Default value: 1.0</td>
</tr>
<tr>
<td>huber_delta</td>
<td>The parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than delta and L1 loss for errors larger than delta. <strong>Optional</strong> Valid values: Positive floating-point integer Default value: 1.0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>init_bias</td>
<td>Initial weight for the bias term.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>init_method</td>
<td>Sets the initial distribution function used for model weights. Functions include:</td>
</tr>
<tr>
<td></td>
<td>• uniform—Uniformly distributed between (-scale, +scale)</td>
</tr>
<tr>
<td></td>
<td>• normal—Normal distribution, with mean 0 and sigma</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform or normal</td>
</tr>
<tr>
<td></td>
<td>Default value: uniform</td>
</tr>
<tr>
<td>init_scale</td>
<td>Scales an initial uniform distribution for model weights. Applies only when the init_method hyperparameter is set to uniform.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.07</td>
</tr>
<tr>
<td>init_sigma</td>
<td>The initial standard deviation for the normal distribution. Applies only when the init_method hyperparameter is set to normal.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</td>
</tr>
<tr>
<td>l1</td>
<td>The L1 regularization parameter. If you don’t want to use L1 regularization, set the value to 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or non-negative float</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The step size used by the optimizer for parameter updates.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto, whose value depends on the optimizer chosen.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| loss                  | Specifies the loss function. The available loss functions and their default values depend on the value of `predictor_type`:

- If the `predictor_type` is set to `regressor`, the available options are `auto`, `squared_loss`, `absolute_loss`, `eps_insensitive_squared_loss`, `eps_insensitive_absolute_loss`, `quantile_loss`, and `huber_loss`. The default value for `auto` is `squared_loss`.
- If the `predictor_type` is set to `binary_classifier`, the available options are `auto`, `logistic`, and `hinge_loss`. The default value for `auto` is `logistic`.
- If the `predictor_type` is set to `multiclass_classifier`, the available options are `auto` and `softmax_loss`. The default value for `auto` is `softmax_loss`.

Valid values: `auto`, `logistic`, `squared_loss`, `absolute_loss`, `hinge_loss`, `eps_insensitive_squared_loss`, `eps_insensitive_absolute_loss`, `quantile_loss`, or `huber_loss`  
Optimal  
Default value: `auto` |
| loss_insensitivity    | The parameter for the epsilon-insensitive loss type. During training and metric evaluation, any error smaller than this value is considered to be zero.  
Optional  
Valid values: Positive floating-point integer  
Default value: 0.01 |
| lr_scheduler_factor   | For every `lr_scheduler_step` hyperparameter, the learning rate decreases by this quantity. Applies only when the `use_lr_scheduler` hyperparameter is set to true.  
Optional  
Valid values: `auto` or positive floating-point integer between 0 and 1  
Default value: `auto` |
| lr_scheduler_minimum_lr | The learning rate never decreases to a value lower than the value set for `lr_scheduler_minimum_lr`. Applies only when the `use_lr_scheduler` hyperparameter is set to true.  
Optional  
Valid values: `auto` or positive floating-point integer  
Default values: `auto` |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| lr_scheduler_step     | The number of steps between decreases of the learning rate. Applies only when the use_lr_scheduler hyperparameter is set to true.  
  **Optional**  
  Valid values: auto or positive integer  
  Default value: auto                                                                 |
| margin                | The margin for the hinge_loss function.  
  **Optional**  
  Valid values: Positive floating-point integer  
  Default value: 1.0                                                                               |
| mini_batch_size       | The number of observations per mini-batch for the data iterator.  
  **Optional**  
  Valid values: Positive integer  
  Default value: 1000                                                                                 |
| momentum              | The momentum of the sgd optimizer.  
  **Optional**  
  Valid values: auto or a floating-point integer between 0 and 1.0  
  Default value: auto                                                                                  |
| normalize_data        | Normalizes the feature data before training. Data normalization shifts the data for each feature to have a mean of zero and scales it to have unit standard deviation.  
  **Optional**  
  Valid values: auto, true, or false  
  Default value: true                                                                                   |
| normalize_label       | Normalizes the label. Label normalization shifts the label to have a mean of zero and scales it to have unit standard deviation.  
  The auto default value normalizes the label for regression problems but does not for classification problems. If you set the normalize_label hyperparameter to true for classification problems, the algorithm ignores it.  
  **Optional**  
  Valid values: auto, true, or false  
  Default value: auto                                                                                   |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| num_calibration_samples | The number of observations from the validation dataset to use for model calibration (when finding the best threshold).  
Optional  
Valid values: auto or positive integer  
Default value: auto |
| num_models | The number of models to train in parallel. For the default, auto, the algorithm decides the number of parallel models to train. One model is trained according to the given training parameter (regularization, optimizer, loss), and the rest by close parameters.  
Optional  
Valid values: auto or positive integer  
Default values: auto |
| num_point_for_scaler | The number of data points to use for calculating normalization or unbiasing of terms.  
Optional  
Valid values: Positive integer  
Default value: 10,000 |
| optimizer | The optimization algorithm to use.  
Optional  
Valid values:  
• auto—The default value.  
• sgd—Stochastic gradient descent.  
• adam—Adaptive momentum estimation.  
• rmsprop—A gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient.  
Default value: auto. The default setting for auto is adam. |
| positive_example_weight_mult | The weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If you want the algorithm to choose a weight so that errors in classifying negative vs. positive examples have equal impact on training loss, specify balanced. If you want the algorithm to choose the weight that optimizes performance, specify auto.  
Optional  
Valid values: balanced, auto, or a positive floating-point integer  
Default value: 1.0 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantile</td>
<td>The quantile for quantile loss. For quantile q, the model attempts to produce predictions so that the value of true_label is greater than the prediction with probability q.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer between 0 and 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>target_precision</td>
<td>The target precision. If binary_classifier_model_selection_criteria is recall_at_target_precision, then precision is held at this value while recall is maximized.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.8</td>
</tr>
<tr>
<td>target_recall</td>
<td>The target recall. If binary_classifier_model_selection_criteria is precision_at_target_recall, then recall is held at this value while precision is maximized.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.8</td>
</tr>
<tr>
<td>unbias_data</td>
<td>Unbiases the features before training so that the mean is 0. By default, data is unbiased if the use_bias hyperparameter is set to true.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto, true, or false</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>unbias_label</td>
<td>Unbiases labels before training so that the mean is 0. Applies to regression only if the use_bias hyperparameter is set to true.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto, true, or false</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>use_bias</td>
<td>Specifies whether the model should include a bias term, which is the intercept term in the linear equation.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>use_lr_scheduler</td>
<td>Whether to use a scheduler for the learning rate. If you want to use a scheduler, specify true.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>wd</td>
<td>The weight decay parameter, also known as the L2 regularization parameter. If you don't want to use L2 regularization, set the value to 0.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or non-negative floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
</tbody>
</table>

**Tune a Linear Learner Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The linear learner algorithm also has an internal mechanism for tuning hyperparameters separate from the automatic model tuning feature described here. By default, the linear learner algorithm tunes hyperparameters by training multiple models in parallel. When you use automatic model tuning, the linear learner internal tuning mechanism is turned off automatically. This sets the number of parallel models, `num_models`, to 1. The algorithm ignores any value that you set for `num_models`.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

**Metrics Computed by the Linear Learner Algorithm**

The linear learner algorithm reports the metrics in the following table, which are computed during training. Choose one of them as the objective metric. To avoid overfitting, we recommend tuning the model against a validation metric instead of a training metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:objective_loss</td>
<td>The mean value of the objective loss function on the test dataset after the model is trained. By default, the loss is logistic loss for binary classification and squared loss for regression. To set the loss to other types, use the loss hyperparameter.</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:binary_classification_accuracy</td>
<td>The accuracy of the final model on the test dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:binary_f_beta</td>
<td>The F_beta score of the final model on the test dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall.</td>
<td>Maximize</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Optimization Direction</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>test:precision</td>
<td>The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:recall</td>
<td>The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:objective</td>
<td>The mean value of the objective loss function on the validation dataset every epoch. By default, the loss is logistic loss for binary classification and squared loss for regression. To set loss to other types, use the <code>loss</code> hyperparameter.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:binary_classification_accuracy</td>
<td>The accuracy of the final model on the validation dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:binary_f_beta</td>
<td>The F_beta score of the final model on the validation dataset. By default, the F_beta score is the F1 score, which is the harmonic mean of the <code>validation:precision</code> and <code>validation:recall</code> metrics.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:precision</td>
<td>The precision of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:recall</td>
<td>The recall of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

**Tuning Linear Learner Hyperparameters**

You can tune a linear learner model with the following hyperparameters.
## Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>wd</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-7, MaxValue: 1</td>
</tr>
<tr>
<td>l1</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-7, MaxValue: 1</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-5, MaxValue: 1</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 100, MaxValue: 5000</td>
</tr>
<tr>
<td>use_bias</td>
<td>CategoricalParameterRanges</td>
<td>[True, False]</td>
</tr>
<tr>
<td>positive_example_weight</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-5, MaxValue: 1e5</td>
</tr>
</tbody>
</table>

### Linear Learner Response Formats

#### JSON Response Format

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. The following are the available output formats for the Amazon SageMaker linear learner algorithm.

#### Binary Classification

```json
let response = {
  predictions: [
    {
      score: 0.4,
      predicted_label: 0
    }
  ]
}
```

#### Multiclass Classification

```json
let response = {
  predictions: [
    {
      score: [0.1, 0.2, 0.4, 0.3],
      predicted_label: 2
    }
  ]
}
```

#### Regression

```json
let response = {
  predictions: [
    {
      score: 0.4
    }
  ]
}
```
JSONLINES Response Format

Binary Classification

```json
{"score": 0.4, "predicted_label": 0}
```

Multiclass Classification

```json
{"score": [0.1, 0.2, 0.4, 0.3], "predicted_label": 2}
```

Regression

```json
{"score": 0.4}
```

RECORDIO Response Format

Binary Classification

```json
[ Record = {
  "features": [],
  "label": {
    "score": {
      "keys": [],
      "values": [0.4]  # float32
    },
    "predicted_label": {
      "keys": [],
      "values": [0.0]  # float32
    }
  }
}
```

Multiclass Classification

```json
[ Record = {
  "features": [],
  "label": {
    "score": {
      "values": [0.1, 0.2, 0.3, 0.4]
    },
    "predicted_label": {
      "values": [3]
    }
  },
  "uid": "abc123",
  "metadata": "{created_at: '2017-06-03'}"
}
```

Regression

```json
[ Record = {
  "features": [],
  "label": {
    "score": {
```
Neural Topic Model (NTM) Algorithm

Amazon SageMaker NTM is an unsupervised learning algorithm that is used to organize a corpus of documents into topics that contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as “bike”, “car”, “train”, “mileage”, and “speed” are likely to share a topic on “transportation” for example. Topic modeling can be used to classify or summarize documents based on the topics detected or to retrieve information or recommend content based on topic similarities. The topics from documents that NTM learns are characterized as a latent representation because the topics are inferred from the observed word distributions in the corpus. The semantics of topics are usually inferred by examining the top ranking words they contain. Because the method is unsupervised, only the number of topics, not the topics themselves, are prespecified. In addition, the topics are not guaranteed to align with how a human might naturally categorize documents.

Topic modeling provides a way to visualize the contents of a large document corpus in terms of the learned topics. Documents relevant to each topic might be indexed or searched for based on their soft topic labels. The latent representations of documents might also be used to find similar documents in the topic space. You can also use the latent representations of documents that the topic model learns for input to another supervised algorithm such as a document classifier. Because the latent representations of documents are expected to capture the semantics of the underlying documents, algorithms based in part on these representations are expected to perform better than those based on lexical features alone.

Although you can use both the Amazon SageMaker NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results on the same input data.

For more information on the mathematics behind NTM, see Neural Variational Inference for Text Processing.

Topics
- Input/Output Interface for the NTM Algorithm (p. 343)
- EC2 Instance Recommendation for the NTM Algorithm (p. 344)
- NTM Sample Notebooks (p. 344)
- NTM Hyperparameters (p. 344)
- Tune an NTM Model (p. 347)
- NTM Response Formats (p. 348)

Input/Output Interface for the NTM Algorithm

Amazon SageMaker Neural Topic Model supports four data channels: train, validation, test, and auxiliary. The validation, test, and auxiliary data channels are optional. If you specify any of these optional channels, set the value of the S3DataDistributionType parameter for them to FullyReplicated. If you provide validation data, the loss on this data is logged at every epoch, and the model stops training as soon as it detects that the validation loss is not improving. If you don't provide validation data, the algorithm stops early based on the training data, but this can be less efficient. If you provide test data, the algorithm reports the test loss from the final model.

The train, validation, and test data channels for NTM support both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats. For CSV format, each row must be represented densely with zero counts for words not present in the corresponding document, and have dimension equal to: (number of records) * (vocabulary size). You can use either File mode or Pipe mode to train models on
data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`. The auxiliary channel is used to supply a text file that contains vocabulary. By supplying the vocabulary file, users are able to see the top words for each of the topics printed in the log instead of their integer IDs. Having the vocabulary file also allows NTM to compute the Word Embedding Topic Coherence (WETC) scores, a new metric displayed in the log that captures similarity among the top words in each topic effectively. The `ContentType` for the auxiliary channel is `text/plain`, with each line containing a single word, in the order corresponding to the integer IDs provided in the data. The vocabulary file must be named `vocab.txt` and currently only UTF-8 encoding is supported.

For inference, `text/csv`, `application/json`, `application/jsonlines`, and `application/x-recordio-protobuf` content types are supported. Sparse data can also be passed for `application/json` and `application/x-recordio-protobuf`. NTM inference returns `application/json` or `application/x-recordio-protobuf` predictions, which include the `topic_weights` vector for each observation.

See the blog post and the companion notebook for more details on using the auxiliary channel and the WETC scores. For more information on how to compute the WETC score, see Coherence-Aware Neural Topic Modeling. We used the pairwise WETC described in this paper for the Amazon SageMaker Neural Topic Model.

For more information on input and output file formats, see NTM Response Formats (p. 348) for inference and the NTM Sample Notebooks (p. 344).

**EC2 Instance Recommendation for the NTM Algorithm**

NTM training supports both GPU and CPU instance types. We recommend GPU instances, but for certain workloads, CPU instances may result in lower training costs. CPU instances should be sufficient for inference.

**NTM Sample Notebooks**

For a sample notebook that uses the Amazon SageMaker NTM algorithm to uncover topics in documents from a synthetic data source where the topic distributions are known, see the Introduction to Basic Functionality of NTM. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**NTM Hyperparameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `feature_dim`  | The vocabulary size of the dataset. Required  
|                | Valid values: Positive integer (min: 1, max: 1,000,000) |
| `num_topics`   | The number of required topics. Required  
<p>|                | Valid values: Positive integer (min: 2, max: 1000) |
| <code>batch_norm</code>   | Whether to use batch normalization during training. Optional |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid values: <em>true or false</em></td>
</tr>
<tr>
<td></td>
<td>Default value: <em>false</em></td>
</tr>
<tr>
<td><strong>clip_gradient</strong></td>
<td>The maximum magnitude for each gradient component.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 1e-3)</td>
</tr>
<tr>
<td></td>
<td>Default value: Infinity</td>
</tr>
<tr>
<td><strong>encoder_layers</strong></td>
<td>The number of layers in the encoder and the output size of each layer. When set to <em>auto</em>, the algorithm uses two layers of sizes 3 x num_topics and 2 x num_topics respectively.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Comma-separated list of positive integers or <em>auto</em></td>
</tr>
<tr>
<td></td>
<td>Default value: <em>auto</em></td>
</tr>
<tr>
<td><strong>encoder_layers_activation</strong></td>
<td>The activation function to use in the encoder layers.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• sigmoid: Sigmoid function</td>
</tr>
<tr>
<td></td>
<td>• tanh: Hyperbolic tangent</td>
</tr>
<tr>
<td></td>
<td>• relu: Rectified linear unit</td>
</tr>
<tr>
<td></td>
<td>Default value: sigmoid</td>
</tr>
<tr>
<td><strong>epochs</strong></td>
<td>The maximum number of passes over the training data.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1)</td>
</tr>
<tr>
<td></td>
<td>Default value: 50</td>
</tr>
<tr>
<td><strong>learning_rate</strong></td>
<td>The learning rate for the optimizer.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 1e-6, max: 1.0)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td><strong>mini_batch_size</strong></td>
<td>The number of examples in each mini batch.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1, max: 10000)</td>
</tr>
<tr>
<td></td>
<td>Default value: 256</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>num_patience_epochs</code></td>
<td>The number of successive epochs over which early stopping criterion is evaluated. Early stopping is triggered when the change in the loss function drops below the specified tolerance within the last <code>num_patience_epochs</code> number of epochs. To disable early stopping, set <code>num_patience_epochs</code> to a value larger than epochs.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1)</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td><code>optimizer</code></td>
<td>The optimizer to use for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
</tbody>
</table>
|                        | • `sgd`: Stochastic gradient descent  
|                        | • `adam`: Adaptive momentum estimation  
|                        | • `adagrad`: Adaptive gradient algorithm  
|                        | • `adadelta`: An adaptive learning rate algorithm  
|                        | • `rmsprop`: Root mean square propagation                                                                                                                                                                   |
|                        | Default value: `adadelta`                                                                                                                                                                                   |
| `rescale_gradient`     | The rescale factor for gradient.                                                                                                                                                                           |
|                        | **Optional**                                                                                                                                                                                                |
|                        | Valid values: float (min: 1e-3, max: 1.0)                                                                                                                                                                  |
|                        | Default value: 1.0                                                                                                                                                                                            |
| `sub_sample`           | The fraction of the training data to sample for training per epoch.                                                                                                                                          |
|                        | **Optional**                                                                                                                                                                                                |
|                        | Valid values: Float (min: 0.0, max: 1.0)                                                                                                                                                                   |
|                        | Default value: 1.0                                                                                                                                                                                            |
| `tolerance`            | The maximum relative change in the loss function. Early stopping is triggered when change in the loss function drops below this value within the last `num_patience_epochs` number of epochs.                              |
|                        | **Optional**                                                                                                                                                                                                |
|                        | Valid values: Float (min: 1e-6, max: 0.1)                                                                                                                                                                 |
|                        | Default value: 0.001                                                                                                                                                                                          |
Parameter Name | Description |
--- | --- |
weight_decay | The weight decay coefficient. Adds L2 regularization. Optional |
| Valid values: Float (min: 0.0, max: 1.0) |
| Default value: 0.0 |

Tune an NTM Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. Latent representations use inferred variables that are not directly measured to model the observations in a dataset. Automatic model tuning on NTM helps you find the model that minimizes loss over the training or validation data. *Training loss* measures how well the model fits the training data. *Validation loss* measures how well the model can generalize to data that it is not trained on. Low training loss indicates that a model is a good fit to the training data. Low validation loss indicates that a model has not overfit the training data and so should be able to model documents on which it has not been trained successfully. Usually, it's preferable to have both losses be small. However, minimizing training loss too much might result in overfitting and increase validation loss, which would reduce the generality of the model.

For more information about model tuning, see [*Perform Automatic Model Tuning*](p. 555).

Metrics Computed by the NTM Algorithm

The NTM algorithm reports a single metric that is computed during training: `validation:total_loss`. The total loss is the sum of the reconstruction loss and Kullback-Leibler divergence. When tuning hyperparameter values, choose this metric as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:total_loss</td>
<td>Total Loss on validation set</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Tunable NTM Hyperparameters

You can tune the following hyperparameters for the NTM algorithm. Usually setting low `mini_batch_size` and small `learning_rate` values results in lower validation losses, although it might take longer to train. Low validation losses don't necessarily produce more coherent topics as interpreted by humans. The effect of other hyperparameters on training and validation loss can vary from dataset to dataset. To see which values are compatible, see [*NTM Hyperparameters*](p. 344).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>encoder_layers_activation</code></td>
<td>CategoricalParameterRanges</td>
<td>['sigmoid', 'tanh', 'relu']</td>
</tr>
<tr>
<td><code>learning_rate</code></td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-4, MaxValue: 0.1</td>
</tr>
</tbody>
</table>
### Parameter Name | Parameter Type | Recommended Ranges
---|---|---
mini_batch_size | IntegerParameterRanges | MinValue: 16, MaxValue: 2048
optimizer | CategoricalParameterRanges | ['sgd', 'adam', 'adadelta']
rescale_gradient | ContinuousParameterRange | MinValue: 0.1, MaxValue: 1.0
weight_decay | ContinuousParameterRange | MinValue: 0.0, MaxValue: 1.0

#### NTM Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker NTM algorithm.

**JSON Response Format**

```json
{
    "predictions": [
        {
            "topic_weights": [0.02, 0.1, 0,...]
        },
        {
            "topic_weights": [0.25, 0.067, 0,...]
        }
    ]
}
```

**JSONLINES Response Format**

```json
{"topic_weights": [0.02, 0.1, 0,...]}
{"topic_weights": [0.25, 0.067, 0,...]}
```

**RECORDIO Response Format**

```json
[  Record = {
    features = {},
    label = {
      'topic_weights': {
        keys: [],
        values: [0.25, 0.067, 0, ...]  # float32
      }
    }
  },
  Record = {
    features = {},
    label = {
      'topic_weights': {
        keys: [],
        values: [0.25, 0.067, 0, ...]  # float32
      }
    }
  }
]
```
Object2Vec Algorithm

The Amazon SageMaker Object2Vec algorithm is a general-purpose neural embedding algorithm that is highly customizable. It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in a way that preserves the semantics of the relationship between pairs of objects in the original space in the embedding space. You can use the learned embeddings to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space, for example. You can also use the embeddings as features of the corresponding objects in downstream supervised tasks, such as classification or regression.

Object2Vec generalizes the well-known Word2Vec embedding technique for words that is optimized in the Amazon SageMaker BlazingText Algorithm (p. 238). For a blog post that discusses how to apply Object2Vec to some practical use cases, see Introduction to Amazon SageMaker Object2Vec.

Topics

- I/O Interface for the Object2Vec Algorithm (p. 349)
- EC2 Instance Recommendation for the Object2Vec Algorithm (p. 350)
- Object2Vec Sample Notebooks (p. 350)
- How Object2Vec Works (p. 351)
- Object2Vec Hyperparameters (p. 352)
- Tune an Object2Vec Model (p. 360)
- Data Formats for Object2Vec Training (p. 362)
- Data Formats for Object2Vec Inference (p. 362)
- Encoder Embeddings for Object2Vec (p. 364)

I/O Interface for the Object2Vec Algorithm

You can use Object2Vec on many input data types, including the following examples.

<table>
<thead>
<tr>
<th>Input Data Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence-sentence pairs</td>
<td>&quot;A soccer game with multiple males playing.&quot; and &quot;Some men are playing a sport.&quot;</td>
</tr>
<tr>
<td>Labels-sequence pairs</td>
<td>The genre tags of the movie &quot;Titanic&quot;, such as &quot;Romance&quot; and &quot;Drama&quot;, and its short description: &quot;James Cameron's Titanic is an epic, action-packed romance set against the ill-fated maiden voyage of the R.M.S. Titanic. She was the most luxurious liner of her era, a ship of dreams, which ultimately carried over 1,500 people to their death in the ice cold waters of the North Atlantic in the early hours of April 15, 1912.&quot;</td>
</tr>
<tr>
<td>Customer-customer pairs</td>
<td>The customer ID of Jane and customer ID of Jackie.</td>
</tr>
<tr>
<td>Product-product pairs</td>
<td>The product ID of football and product ID of basketball.</td>
</tr>
<tr>
<td>Item review user-item pairs</td>
<td>A user's ID and the items she has bought, such as apple, pear, and orange.</td>
</tr>
</tbody>
</table>

To transform the input data into the supported formats, you must preprocess it. Currently, Object2Vec natively supports two types of input:

- A discrete token, which is represented as a list of a single integer-id. For example, [10].
A sequences of discrete tokens, which is represented as a list of integer-ids. For example, 
[0,12,10,13].

The object in each pair can be asymmetric. For example, the pairs can be (token, sequence) or (token, 
token) or (sequence, sequence). For token inputs, the algorithm supports simple embeddings as 
compatible encoders. For sequences of token vectors, the algorithm supports the following as encoders:

• Average-pooled embeddings
• Hierarchical convolutional neural networks (CNNs),
• Multi-layered bidirectional long short-term memory (BiLSTMs)

The input label for each pair can be one of the following:

• A categorical label that expresses the relationship between the objects in the pair
• A score that expresses the strength of the similarity between the two objects

For categorical labels used in classification, the algorithm supports the cross-entropy loss function. For 
ratings/score-based labels used in regression, the algorithm supports the mean squared error (MSE) loss 
function. Specify these loss functions with the output_layer hyperparameter when you create the 
model training job.

EC2 Instance Recommendation for the Object2Vec Algorithm

The type of Amazon Elastic Compute Cloud (Amazon EC2) instance that you use depends on whether you 
are training or running inferences.

Instance Recommendation for Training

When training a model using the Object2Vec algorithm on a CPU, start with an ml.m5.2xlarge instance. 
For training on a GPU, start with an ml.p2.xlarge instance. If the training takes too long on this instance, 
you can use a larger instance, such as an ml.m5.4xlarge or an ml.m5.12xlarge instance Currently, the 
Object2Vec algorithm can train only on a single machine. However, it does offer support for multiple 
GPUs.

Instance Recommendation for Inference

For inference with a trained Object2Vec model that has a deep neural network, we recommend 
using ml.p3.2xlarge GPU instance. Due to GPU memory scarcity, the INFERENECE_PREFERRED_MODE 
environment variable can be specified to optimize on whether the section called “GPU 
optimization: Classification or Regression” (p. 362) or the section called “GPU optimization: Encoder 
Embeddings” (p. 364) inference network is loaded into GPU.

Object2Vec Sample Notebooks

For a sample notebook that uses the Amazon SageMaker Object2Vec algorithm to encode sequences 
into fixed-length embeddings, see Using Object2Vec to Encode Sentences into Fixed Length 
Embeddings. For a sample notebook that uses the Object2Vec algorithm in a multi-label prediction 
setting to predict the genre of a movie from its plot description, see Movie genre prediction 
with Object2Vec Algorithm. For a sample notebook that uses the Object2Vec algorithm to make 
movie recommendations, see An Introduction to SageMaker ObjectToVec model for MovieLens 
recommendation. For a sample notebook that uses the Object2Vec algorithm to learn document 
embeddings, see Using Object2Vec to learn document embeddings. For instructions on how to create 
and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see 
Use Amazon SageMaker Notebook Instances (p. 201). After you have created a notebook instance
and opened it, choose SageMaker Examples to see a list of Amazon SageMaker samples. To open a notebook, choose its Use tab and choose Create copy.

How Object2Vec Works

When using the Amazon SageMaker Object2Vec algorithm, you follow the standard workflow: process the data, train the model, and produce inferences.

Topics

- Step 1: Process Data (p. 351)
- Step 2: Train a Model (p. 351)
- Step 3: Produce Inferences (p. 352)

Step 1: Process Data

During preprocessing, convert the data to the JSON Lines text file format specified in Data Formats for Object2Vec Training (p. 362). To get the highest accuracy during training, also randomly shuffle the data before feeding it into the model. How you generate random permutations depends on the language. For python, you could use np.randon.shuffle; for Unix, shuf.

Step 2: Train a Model

The Amazon SageMaker Object2Vec algorithm has the following main components:

- Two input channels – The input channels take a pair of objects of the same or different types as inputs, and pass them to independent and customizable encoders.
- Two encoders – The two encoders, enc0 and enc1, convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair are then passed into a comparator.
- A comparator – The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects. In the output score for a sentence pair. For example, 1 indicates a strong relationship between a sentence pair, and 0 represents a weak relationship.

During training, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types, as described earlier. If the inputs to both encoders are composed of the same token-level units, you can use a shared token embedding layer by setting the tied_token_embedding_weight hyperparameter to True when you create the training job. This is possible, for example, when comparing sentences that both have word token-level units. To generate negative samples at a specified rate, set the negative_sampling_rate hyperparameter to the desired ratio of negative to positive samples. This hyperparameter expedites learning how to discriminate between the positive samples observed in the training data and the negative samples that are not likely to be observed.

Pairs of objects are passed through independent, customizable encoders that are compatible with the input types of corresponding objects. The encoders convert each object in a pair into a fixed-length embedding vector of equal length. The pair of vectors are passed to a comparator operator, which assembles the vectors into a single vector using the value specified in the he comparator_list hyperparameter. The assembled vector then passes through a multilayer perceptron (MLP) layer, which produces an output that the loss function compares with the labels that you provided. This comparison evaluates the strength of the relationship between the objects in the pair as predicted by the model. The following figure shows this workflow.
Step 3: Produce Inferences

After the model is trained, you can use the trained encoder to preprocess input objects or to perform two types of inference:

- To convert singleton input objects into fixed-length embeddings using the corresponding encoder
- To predict the relationship label or score between a pair of input objects

The inference server automatically figures out which of the types is requested based on the input data. To get the embeddings as output, provide only one input. To predict the relationship label or score, provide both inputs in the pair.

Object2Vec Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Object2Vec training algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enc0_max_seq_len</code></td>
<td>The maximum sequence length for the enc0 encoder.</td>
</tr>
<tr>
<td>Required</td>
<td>Valid values: $1 \leq integer \leq 5000$</td>
</tr>
<tr>
<td><code>enc0_vocab_size</code></td>
<td>The vocabulary size of enc0 tokens.</td>
</tr>
<tr>
<td>Required</td>
<td>Valid values: $2 \leq integer \leq 3000000$</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bucket_width</td>
<td>The allowed difference between data sequence length when bucketing is enabled. To enable bucketing, specify a non-zero value for this parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Valid values:</strong> $0 \leq \text{integer} \leq 100$</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 0 (no bucketing)</td>
</tr>
<tr>
<td>comparator_list</td>
<td>A list used to customize the way in which two embeddings are compared. The Object2Vec comparator operator layer takes the encodings from both encoders as inputs and outputs a single vector. This vector is a concatenation of subvectors. The string values passed to the comparator_list and the order in which they are passed determine how these subvectors are assembled. For example, if comparator_list=&quot;hadamard, concat&quot;, then the comparator operator constructs the vector by concatenating the Hadamard product of two encodings and the concatenation of two encodings. If, on the other hand, comparator_list=&quot;hadamard&quot;, then the comparator operator constructs the vector as the hadamard product of only two encodings.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Valid values:</strong> A string that contains any combination of the names of the three binary operators: hadamard, concat, or abs_diff. The Object2Vec algorithm currently requires that the two vector encodings have the same dimension. These operators produce the subvectors as follows:</td>
</tr>
<tr>
<td></td>
<td>• hadamard: Constructs a vector as the Hadamard (element-wise) product of two encodings.</td>
</tr>
<tr>
<td></td>
<td>• concat: Constructs a vector as the concatenation of two encodings.</td>
</tr>
<tr>
<td></td>
<td>• abs_diff: Constructs a vector as the absolute difference between two encodings.</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> &quot;hadamard, concat, abs_diff&quot;</td>
</tr>
<tr>
<td>dropout</td>
<td>The dropout probability for network layers. Dropout is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Valid values:</strong> $0.0 \leq \text{float} \leq 1.0$</td>
</tr>
<tr>
<td></td>
<td><strong>Default value:</strong> 0.0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by with the early_stopping_tolerance hyperparameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 5$</td>
</tr>
<tr>
<td></td>
<td>Default value: $3$</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after the number of consecutive epochs specified in the early_stopping_patience hyperparameter concludes.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0.000001 \leq \text{float} \leq 0.1$</td>
</tr>
<tr>
<td></td>
<td>Default value: $0.01$</td>
</tr>
<tr>
<td>enc_dim</td>
<td>The dimension of the output of the embedding layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $4 \leq \text{integer} \leq 10000$</td>
</tr>
<tr>
<td></td>
<td>Default value: $4096$</td>
</tr>
<tr>
<td>enc0_network</td>
<td>The network model for the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: hcnn, bilstm, or pooled_embedding</td>
</tr>
<tr>
<td></td>
<td>• hcnn: A hierarchical convolutional neural network.</td>
</tr>
<tr>
<td></td>
<td>• bilstm: A bidirectional long short-term memory network (biLSTM), in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks.</td>
</tr>
<tr>
<td></td>
<td>• pooled_embedding: Averages the embeddings of all of the tokens in the input.</td>
</tr>
<tr>
<td></td>
<td>Default value: hcnn</td>
</tr>
<tr>
<td>enc0_cnn_filter_width</td>
<td>The filter width of the convolutional neural network (CNN) enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 9$</td>
</tr>
<tr>
<td></td>
<td>Default value: $3$</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enc0_freeze_pretrained_embedding</td>
<td>Whether to freeze enc0 pretrained embedding weights.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: True</td>
</tr>
<tr>
<td>enc0_layers</td>
<td>The number of layers in the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or (1 \leq \text{integer} \leq 4)</td>
</tr>
<tr>
<td></td>
<td>- For hcnn, auto means 4.</td>
</tr>
<tr>
<td></td>
<td>- For bilstm, auto means 1.</td>
</tr>
<tr>
<td></td>
<td>- For pooled_embedding, auto ignores the number of layers.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>enc0_pretrained_embedding_file</td>
<td>The filename of the pretrained enc0 token embedding file in the auxiliary data channel.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9._]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>enc0_token_embedding_dim</td>
<td>The output dimension of the enc0 token embedding layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: (2 \leq \text{integer} \leq 1000)</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
<tr>
<td>enc0_vocab_file</td>
<td>The vocabulary file for mapping pretrained enc0 token embedding vectors to numerical vocabulary IDs.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9._]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enc1_network</td>
<td>The network model for the enc1 encoder. If you want the enc1 encoder to use the same network model as enc0, including the hyperparameter values, set the value to enc0.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Even when the enc0 and enc1 encoder networks have symmetric architectures, you can't shared parameter values for these networks.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: enc0, hcnn, bilstm, or pooled_embedding</td>
</tr>
<tr>
<td></td>
<td>• enc0: The network model for the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td>• hcnn: A hierarchical convolutional neural network.</td>
</tr>
<tr>
<td></td>
<td>• bilstm: A bidirectional LSTM, in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks.</td>
</tr>
<tr>
<td></td>
<td>• pooled_embedding: The averages of the embeddings of all of the tokens in the input.</td>
</tr>
<tr>
<td></td>
<td>Default value: enc0</td>
</tr>
<tr>
<td>enc1_cnn_filter_width</td>
<td>The filter width of the CNN enc1 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 9</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>enc1_freeze_pretrained_embedding</td>
<td>Whether to freeze enc1 pretrained embedding weights.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: True</td>
</tr>
<tr>
<td>enc1_layers</td>
<td>The number of layers in the enc1 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or $1 \leq \text{integer} \leq 4</td>
</tr>
<tr>
<td></td>
<td>• For hcnn, auto means 4.</td>
</tr>
<tr>
<td></td>
<td>• For bilstm, auto means 1.</td>
</tr>
<tr>
<td></td>
<td>• For pooled_embedding, auto ignores the number of layers.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>enc1_max_seq_len</td>
<td>The maximum sequence length for the enc1 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 5000</td>
</tr>
</tbody>
</table>

356
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enc1_pretrained_embedding_file</td>
<td>The name of the enc1 pretrained token embedding file in the auxiliary data channel.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9._]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>enc1_token_embedding_dim</td>
<td>The output dimension of the enc1 token embedding layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 2 ≤ integer ≤ 1000</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
<tr>
<td>enc1_vocab_file</td>
<td>The vocabulary file for mapping pretrained enc1 token embeddings to vocabulary IDs.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9._]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>enc1_vocab_size</td>
<td>The vocabulary size of enc0 tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 2 ≤ integer ≤ 3000000</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of epochs to run for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 ≤ integer ≤ 100</td>
</tr>
<tr>
<td></td>
<td>Default value: 30</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 1.0E-6 ≤ float ≤ 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0004</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size that the dataset is split into for an optimizer during training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 ≤ integer ≤ 10000</td>
</tr>
<tr>
<td></td>
<td>Default value: 32</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mlp_activation</td>
<td>The type of activation function for the multilayer perceptron (MLP) layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>tanh</code>, <code>relu</code>, or <code>linear</code></td>
</tr>
<tr>
<td></td>
<td>- <code>tanh</code>: Hyperbolic tangent</td>
</tr>
<tr>
<td></td>
<td>- <code>relu</code>: Rectified linear unit (ReLU)</td>
</tr>
<tr>
<td></td>
<td>- <code>linear</code>: Linear function</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>linear</code></td>
</tr>
<tr>
<td>mlp_dim</td>
<td>The dimension of the output from MLP layers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 10000$</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td>mlp_layers</td>
<td>The number of MLP layers in the network.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{integer} \leq 10$</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>negative_sampling_rate</td>
<td>The ratio of negative samples, generated to assist in training the algorithm, to positive samples that are provided by users. Negative samples represent data that is unlikely to occur in reality and are labelled negatively for training. They facilitate training a model to discriminate between the positive samples observed and the negative samples that are not. To specify the ratio of negative samples to positive samples used for training, set the value to a positive integer. For example, if you train the algorithm on input data in which all of the samples are positive and set <code>negative_sampling_rate</code> to 2, the Object2Vec algorithm internally generates two negative samples per positive sample. If you don’t want to generate or use negative samples during training, set the value to 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{integer}$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0 (off)</td>
</tr>
<tr>
<td>num_classes</td>
<td>The number of classes for classification training. Amazon SageMaker ignores this hyperparameter for regression problems.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 30$</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer type.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>adadelta</code>, <code>adagrad</code>, <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</td>
</tr>
<tr>
<td></td>
<td>• <code>adadelta</code>: A <em>per-dimension learning rate method</em> for gradient descent</td>
</tr>
<tr>
<td></td>
<td>• <code>adagrad</code>: The <em>adaptive gradient algorithm</em></td>
</tr>
<tr>
<td></td>
<td>• <code>adam</code>: The <em>adaptive moment estimation algorithm</em></td>
</tr>
<tr>
<td></td>
<td>• <code>sgd</code>: <em>Stochastic gradient descent</em></td>
</tr>
<tr>
<td></td>
<td>• <code>rmsprop</code>: <em>Root mean square propagation</em></td>
</tr>
<tr>
<td></td>
<td>Default value: <code>adam</code></td>
</tr>
<tr>
<td>output_layer</td>
<td>The type of output layer where you specify that the task is regression or classification.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>softmax</code> or <code>mean_squared_error</code></td>
</tr>
<tr>
<td></td>
<td>• <code>softmax</code>: The <em>Softmax function</em> used for classification.</td>
</tr>
<tr>
<td></td>
<td>• <code>mean_squared_error</code>: The <em>MSE</em> used for regression.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>softmax</code></td>
</tr>
<tr>
<td>tied_token_embedding_weight</td>
<td>Whether to use a shared embedding layer for both encoders. If the inputs to both encoders use the same token-level units, use a shared token embedding layer. For example, for a collection of documents, if one encoder encodes sentences and another encodes whole documents, you can use a shared token embedding layer. That's because both sentences and documents are composed of word tokens from the same vocabulary.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>True</code> or <code>False</code></td>
</tr>
<tr>
<td></td>
<td>Default value: <code>False</code></td>
</tr>
</tbody>
</table>
### Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| token_embedding_storage_type | The mode of gradient update used during training; when the dense mode is used, the optimizer calculates the full gradient matrix for the token embedding layer even if most rows of the gradient are zero-valued. When sparse mode is used, the optimizer only stores rows of the gradient that are actually being used in the mini-batch. If you want the algorithm to perform lazy gradient updates, which calculate the gradients only in the non-zero rows and which speed up training, specify row_sparse. Setting the value to row_sparse constrains the values available for other hyperparameters, as follows:  
  - The optimizer hyperparameter must be set to adam, adagrad, or sgd. Otherwise, the algorithm throws a CustomerValueError.  
  - The algorithm automatically disables bucketing, setting the bucket_width hyperparameter to 0.  
  
  **Optional**  
  Valid values: dense or row_sparse  
  Default value: dense |
| weight_decay | The weight decay parameter used for optimization.  
  
  **Optional**  
  Valid values: 0 ≤ float ≤ 10000  
  Default value: 0 (no decay) |

### Tune an Object2Vec Model

**Automatic model tuning**, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. For the objective metric, you use one of the metrics that the algorithm computes. Automatic model tuning searches the chosen hyperparameters to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

**Metrics Computed by the Object2Vec Algorithm**

The Object2Vec algorithm has both classification and regression metrics. The output_layer type determines which metric you can use for automatic model tuning.

**Regressor Metrics Computed by the Object2Vec Algorithm**

The algorithm reports a mean squared error regressor metric, which is computed during testing and validation. When tuning the model for regression tasks, choose this metric as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:mean_squared_error</td>
<td>Mean Square Error</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
### Classification Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm reports accuracy and cross-entropy classification metrics, which are computed during test and validation. When tuning the model for classification tasks, choose one of these as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:mean_squared_error</td>
<td>Root Mean Square Error</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:cross_entropy</td>
<td>Cross-entropy</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:cross_entropy</td>
<td>Cross-entropy</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

### Tunable Object2Vec Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

<table>
<thead>
<tr>
<th>Hyperparameter Name</th>
<th>Hyperparameter Type</th>
<th>Recommended Ranges and Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropout</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
<tr>
<td>early_stopping_pat</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>early_stopping_tol</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.001, MaxValue: 0.1</td>
</tr>
<tr>
<td>enc_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 4, MaxValue: 4096</td>
</tr>
<tr>
<td>enc0_cnn_filter_w</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>enc0_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>enc0_token_embedding</td>
<td>IntegerParameterRange</td>
<td>MinValue: 5, MaxValue: 300</td>
</tr>
<tr>
<td>enc1_cnn_filter_w</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>enc1_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>enc1_token_embedding</td>
<td>IntegerParameterRange</td>
<td>MinValue: 5, MaxValue: 300</td>
</tr>
<tr>
<td>Hyperparameter Name</td>
<td>Hyperparameter Type</td>
<td>Recommended Ranges and Values</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>MinValue: 4, MaxValue: 20</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-6, MaxValue: 1.0</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 8192</td>
</tr>
<tr>
<td>mlp_activation</td>
<td>CategoricalParameterRanges</td>
<td>[tanh, relu, linear]</td>
</tr>
<tr>
<td>mlp_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 16, MaxValue: 1024</td>
</tr>
<tr>
<td>mlp_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>[adagrad, adam, rmsprop, sgd, adadelta]</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
</tbody>
</table>

Data Formats for Object2Vec Training

Input: JSON Lines Request Format

Content-type: application/jsonlines

```
{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}
{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}
{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

The “in0” and “in1” are the inputs for encoder0 and encoder1, respectively. The same format is valid for both classification and regression problems. For regression, the field "label" can accept real valued inputs.

Data Formats for Object2Vec Inference

GPU optimization: Classification or Regression

Due to GPU memory scarcity, the INFERENCE_PREFERRED_MODE environment variable can be specified to optimize on whether the classification/regression or the section called “Output: Encoder Embeddings” (p. 364) inference network is loaded into GPU. If the majority of your inference is for classification or regression, specify INFERENCE_PREFERRED_MODE=classification. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for classification/regression inference:

```python
transformer = o2v.transformer(instance_count=4,
                               instance_type="ml.p2.xlarge",
                               max_concurrent_transforms=2,
                               INFERENCE_PREFERRED_MODE="classification")
```
Use Built-in Algorithms

Input: Classification or Regression Request Format

Content-type: application/json

```
{  
  "instances" : [  
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4],   "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]},  
    {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4],   "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]},  
    {"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}  
  ]
}
```

Content-type: application/json

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}  
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}  
{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

For classification problems, the length of the scores vector corresponds to `num_classes`. For regression problems, the length is 1.

Output: Classification or Regression Response Format

Accept: application/json

```
{  
  "predictions": [  
    {  
      "scores": [  
        0.6533935070037842,  
        0.07582679390907288,  
        0.2707797586917877  
      ],  
      "scores": [  
        0.026291321963071823,  
        0.6577019095420837,  
        0.31600672006607056  
      ]  
    },  
    {  
      "scores": [  
        0.251988261938095,  
        0.258233487606048,  
        0.489778339862823  
      ]  
    },  
    {  
      "scores": [  
        0.280087798833847,  
        0.368331134319305,  
        0.351581096649169  
      ]  
    }  
  ]
}
```

Accept: application/json

```
{"scores":[0.195667684078216,0.3953515558923721,0.408980727195739]}  
{"scores":[0.251988261938095,0.258233487606048,0.489778339862823]}  
{"scores":[0.280087798833847,0.368331134319305,0.351581096649169]}
```

In both the classification and regression formats, the scores apply to individual labels.
Encoder Embeddings for Object2Vec

GPU optimization: Encoder Embeddings

Due to GPU memory scarcity, the `INFERNECE_PREFERRED_MODE` environment variable can be specified to optimize on whether the section called "Inference Formats: Scoring" (p. 362) or the encoder embedding inference network is loaded into GPU. If the majority of your inference is for encoder embeddings, specify `INFERNECE_PREFERRED_MODE=embedding`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for encoder embedding inference:

```python
transformer = o2v.transformer(instance_count=4,
    instance_type="ml.p2.xlarge",
    max_concurrent_transforms=2,
    max_payload=1,  # 1MB
    strategy='MultiRecord',
    env={'INFERNECE_PREFERRED_MODE': 'embedding'},  # only useful with GPU
    output_path=output_s3_path)
```

Input: Encoder Embeddings

Content-type: application/json

```
{
    "instances" : [
        {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]},
        {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]},
        {"in0": [774, 14, 21, 206]}
    ]
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]}
{"in0": [774, 14, 21, 206]}
```

In both of these formats, you specify only one input type: “in0” or “in1.” The inference service then invokes the corresponding encoder and outputs the embeddings for each of the instances.

Output: Encoder Embeddings

Content-type: application/json

```
{
    "predictions": [
        {"embeddings":
          [0.057368703186511, 0.030703511089086, 0.099890425801277, 0.063688032329082, 0.026327300816774, 0.003637571120634, 0.021305780857801, 0.004316598642617, 0.0, 0.003397724591195, 0.0, 0.000378780066967, 0.0, 0.0, 0.0, 0.007419463712722]
        },
        {"embeddings":
          [0.150190666317939, 0.05145975202322, 0.098204270005226, 0.064249359071254, 0.056249320507049, 0.01513972133398, 0.047553978860378, 0.0, 0.0, 0.011533712036907, 0.011472506448626, 0.010696629062294, 0.0, 0.0, 0.0, 0.008508535102009]
        }
    ]
}
```

Content-type: application/jsonlines

```
{"embeddings":
  [0.057368703186511, 0.030703511089086, 0.099890425801277, 0.063688032329082, 0.026327300816774, 0.003637571120634, 0.021305780857801, 0.004316598642617, 0.0, 0.003397724591195, 0.0, 0.000378780066967, 0.0, 0.0, 0.0, 0.007419463712722]}
{"embeddings":
  [0.150190666317939, 0.05145975202322, 0.098204270005226, 0.064249359071254, 0.056249320507049, 0.01513972133398, 0.047553978860378, 0.0, 0.0, 0.011533712036907, 0.011472506448626, 0.010696629062294, 0.0, 0.0, 0.0, 0.008508535102009]}
```
The vector length of the embeddings output by the inference service is equal to the value of one of the following hyperparameters that you specify at training time: `enc0_token_embedding_dim`, `enc1_token_embedding_dim`, or `enc_dim`.

**Object Detection Algorithm**

The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene. The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box. It uses the Single Shot multibox Detector (SSD) framework and supports two base networks: VGG and ResNet. The network can be trained from scratch, or trained with models that have been pre-trained on the ImageNet dataset.

**Topics**
- **Input/Output Interface for the Object Detection Algorithm** (p. 365)
- **EC2 Instance Recommendation for the Object Detection Algorithm** (p. 382)
- **Object Detection Sample Notebooks** (p. 382)
- **How Object Detection Works** (p. 382)
- **Object Detection Hyperparameters** (p. 383)
- **Tune an Object Detection Model** (p. 388)
- **Object Detection Request and Response Formats** (p. 389)

**Input/Output Interface for the Object Detection Algorithm**

The Amazon SageMaker Object Detection algorithm supports both RecordIO (application/x-recordio) and image (image/png, image/jpeg, and application/x-image) content types for training in file mode and supports RecordIO (application/x-recordio) for training in pipe mode. However you can also train in pipe mode using the image files (image/png, image/jpeg, and application/x-image), without creating RecordIO files, by using the augmented manifest format. The recommended input format for the Amazon SageMaker object detection algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format. The algorithm supports only application/x-image for inference.

**Note**
To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

See the **Object Detection Sample Notebooks** (p. 382) for more details on data formats.

**Train with the RecordIO Format**

If you use the RecordIO format for training, specify both train and validation channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` (p. 931) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to application/x-recordio. An example of how to generate RecordIO files can be found in the object detection sample notebook. You can also use tools from the MXNet example to generate RecordIO files for popular datasets like the PASCAL Visual Object Classes and Common Objects in Context (COCO).

**Train with the Image Format**

If you use the image format for training, specify `train`, `validation`, `train_annotation`, and `validation_annotation` channels as values for the `InputDataConfig` parameter of `CreateTrainingJob` (p. 931) request. Specify the individual image data (.jpg or .png) files for the train and validation channels. For annotation data, you can use the JSON format. Specify the corresponding .json files in the `train_annotation` and `validation_annotation` channels. Set the
content type for all four channels to image/png or image/jpeg based on the image type. You can also use the content type application/x-image when your dataset contains both .jpg and .png images. The following is an example of a .json file.

```json
{
  "file": "your_image_directory/sample_image1.jpg",
  "image_size": [
    {
      "width": 500,
      "height": 400,
      "depth": 3
    }
  ],
  "annotations": [
    {
      "class_id": 0,
      "left": 111,
      "top": 134,
      "width": 61,
      "height": 128
    },
    {
      "class_id": 0,
      "left": 161,
      "top": 250,
      "width": 79,
      "height": 143
    },
    {
      "class_id": 1,
      "left": 101,
      "top": 185,
      "width": 42,
      "height": 130
    }
  ],
  "categories": [
    {
      "class_id": 0,
      "name": "dog"
    },
    {
      "class_id": 1,
      "name": "cat"
    }
  ]
}
```

Each image needs a .json file for annotation, and the .json file should have the same name as the corresponding image. The name of above .json file should be "sample_image1.json". There are four properties in the annotation .json file. The property "file" specifies the relative path of the image file. For example, if your training images and corresponding .json files are stored in s3://your_bucket/train/sample_image and s3://your_bucket/train_annotation, specify the path for your train and train_annotation channels as s3://your_bucket/train and s3://your_bucket/train_annotation, respectively.

In the .json file, the relative path for an image named sample_image1.jpg should be sample_image/sample_image1.jpg. The "image_size" property specifies the overall image dimensions. The SageMaker object detection algorithm currently only supports 3-channel images. The "annotations" property specifies the categories and bounding boxes for objects within the image. Each object is annotated by a "class_id" index and by four bounding box coordinates ("left", "top", "width", "height"). The "left" (x-coordinate) and "top" (y-coordinate) values represent the upper-left corner
of the bounding box. The "width" (x-coordinate) and "height" (y-coordinate) values represent the dimensions of the bounding box. The origin (0, 0) is the upper-left corner of the entire image. If you have multiple objects within one image, all the annotations should be included in a single .json file. The "categories" property stores the mapping between the class index and class name. The class indices should be numbered successively and the numbering should start with 0. The "categories" property is optional for the annotation .json file

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the InputDataConfig parameter of the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.

- **HyperParameters** - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

- **InputDataConfig** - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.

- **OutputDataConfig** - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.

- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

- **EnableManagedSpotTraining** - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.

- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.

- **StoppingCondition** - To help cap training costs, use MaxRuntimeInSeconds to set a time limit for training. Use MaxWaitTimeInSeconds to specify how long you are willing to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see How It Works.

### Request Syntax

```json
{
  "AlgorithmSpecification": {
    "AlgorithmName": "string",
    "EnableSageMakerMetricsTimeSeries": boolean,
    "MetricDefinitions": [
      {
        "Name": "string",
      }
    ]
  }
}
```
"Regex": "string"
}
],
"TrainingImage": "string",
"TrainingInputMode": "string"
},
"CheckpointConfig": {
"LocalPath": "string",
"S3Uri": "string"
},
"DebugHookConfig": {
"CollectionConfigurations": [
{
"CollectionName": "string",
"CollectionParameters": {
"string": "string"
}
}
],
"HookParameters": {
"string": "string"
},
"LocalPath": "string",
"S3OutputPath": "string"
},
"DebugRuleConfigurations": [
{
"InstanceType": "string",
"LocalPath": "string",
"RuleConfigurationName": "string",
"RuleEvaluatorImage": "string",
"RuleParameters": {
"string": "string"
The request accepts the following data in JSON format.

AlgorithmSpecification (p. 931)
The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 1274) object
Required: Yes
Type: CheckpointConfig (p. 1314) object

Required: No

**DebugHookConfig (p. 931)**

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

Required: No

**DebugRuleConfigurations (p. 931)**

Configuration information for debugging rules.

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

**EnableInterContainerTrafficEncryption (p. 931)**

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

**EnableManagedSpotTraining (p. 931)**

To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. This option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

**EnableNetworkIsolation (p. 931)**

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

Required: No

**ExperimentConfig (p. 931)**

Configuration for the experiment.
HyperParameters (p. 931)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: .*

Value Length Constraints: Maximum length of 256.

Value Pattern: .*

InputDataConfig (p. 931)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

OutputDataConfig (p. 931)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

Required: Yes

ResourceConfig (p. 931)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the
TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object

Required: Yes

RoleArn (p. 931)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-z-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@-_]/+$

Required: Yes

StoppingCondition (p. 931)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object

Required: Yes

Tags (p. 931)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TensorBoardOutputConfig (p. 931)

Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

TrainingJobName (p. 931)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**VpcConfig (p. 931)**

A VpcConfig (p. 1577) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

**Response Syntax**

```json
{
   "TrainingJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**TrainingJobArn (p. 937)**

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z-]+:sagemaker:[a-z0-9-]+:[0-9]{12}:training-job/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p. ) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the 'source-ref' tag that points to the S3 location of the image. The annotations are provided under the "AttributeNames" parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.
- **HyperParameters** - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- **InputDataConfig** - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- **EnableManagedSpotTraining** - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see Managed Spot Training.
- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- **StoppingCondition** - To help cap training costs, use MaxRuntimeInSeconds to set a time limit for training. Use MaxWaitTimeInSeconds to specify how long you are willing to wait for a managed spot training job to complete.
For more information about Amazon SageMaker, see How It Works.

Request Syntax

```json
{
   "AlgorithmSpecification": {
      "AlgorithmName": "string",
      "EnableSageMakerMetricsTimeSeries": boolean,
      "MetricDefinitions": [
         {
            "Name": "string",
            "Regex": "string"
         }
      ],
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "CheckpointConfig": {
      "LocalPath": "string",
      "S3Uri": "string"
   },
   "DebugHookConfig": {
      "CollectionConfigurations": [
         {
            "CollectionName": "string",
            "CollectionParameters": {
               "string": "string"
            }
         }
      ],
      "HookParameters": {
         "string": "string"
      }
   },
   "LocalPath": "string",
   "S3OutputPath": "string"
}
```

EnableInterContainerTrafficEncryption: boolean, EnableManagedSpotTraining: boolean, EnableNetworkIsolation: boolean, ExperimentConfig: {
   "ExperimentName": "string",
   "TrialComponentDisplayName": "string",
   "TrialName": "string"
},

HyperParameters: {
   "string": "string"
},

InputDataConfig: [{
   "ChannelName": "string",
   "CompressionType": "string",
   "ContentType": "string"
}]
```
"DataSource": {
  "FileSystemDataSource": {
    "DirectoryPath": "string",
    "FileSystemAccessMode": "string",
    "FileSystemId": "string",
    "FileSystemType": "string"
  },
  "S3DataSource": {
    "AttributeNames": [ "string" ],
    "S3DataDistributionType": "string",
    "S3DataType": "string",
    "S3Uri": "string"
  }
},
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
  "Seed": number
}
},
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"Tags": [ {
  "Key": "string",
  "Value": "string"
} ],
"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},
"TrainingJobName": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}
about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 1274) object

Required: Yes

**CheckpointConfig (p. 931)**

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

Required: No

**DebugHookConfig (p. 931)**

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

Required: No

**DebugRuleConfigurations (p. 931)**

Configuration information for debugging rules.

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

**EnableInterContainerTrafficEncryption (p. 931)**

To encrypt all communications between ML compute instances in distributed training, choose `True`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

**EnableManagedSpotTraining (p. 931)**

To train models using managed spot training, choose `True`. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. This option is useful when training jobs can be interrupted and when there is flexibility when the training job is run.

The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No
EnableNetworkIsolation (p. 931)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

Required: No

ExperimentConfig (p. 931)

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

HyperParameters (p. 931)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: . *

Value Length Constraints: Maximum length of 256.

Value Pattern: . *

Required: No

InputDataConfig (p. 931)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No
OutputDataConfig (p. 931)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

Required: Yes

ResourceConfig (p. 931)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object

Required: Yes

RoleArn (p. 931)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/*]+$

Required: Yes

StoppingCondition (p. 931)

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object

Required: Yes

Tags (p. 931)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.
Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TensorBoardOutputConfig (p. 931)

Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

TrainingJobName (p. 931)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9-]\*(-*[a-zA-Z0-9-])*$  

Required: Yes

VpcConfig (p. 931)

A VpcConfig (p. 1577) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

Response Syntax

```json
{
    "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 937)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:training-job/.*
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being accessed is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(request. It can also contain additional metadata under the metadata tag, but these are ignored by the algorithm. In the following example, the "AttributeNames are contained in the list

```json
{"source-ref": "s3://your_bucket/image1.jpg", "bounding-box":{"image_size": [{ "width": 500, "height": 400, "depth":3}]], "annotations": [{"class_id": 0, "left": 111, "top": 134, "width": 61, "height": 128}, {"class_id": 5, "left": 161, "top": 250, "width": 80, "height": 50}], "bounding-box-metadata":{"class-map":{"0": "dog", "5": "horse"}, "type": "groundtruth/object_detection"})
{"source-ref": "s3://your_bucket/image2.jpg", "bounding-box":{"image_size": [{ "width": 400, "height": 300, "depth":3}], "annotations": [{"class_id": 1, "left": 100, "top": 120, "width": 43, "height": 78}], "bounding-box-metadata":{"class-map":{"1": "cat"}, "type": "groundtruth/object_detection"}}
```

The order of "AttributeNames" in the input files matters when training the Object Detection algorithm. It accepts piped data in a specific order, with image first, followed by annotations. So the "AttributeNames" in this example are provided with "source-ref" first, followed by "bounding-box".
When using Object Detection with Augmented Manifest, the value of parameter `RecordWrapperType` must be set as "RecordIO".

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

**Incremental Training**

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. Amazon SageMaker object detection models can be seeded only with another built-in object detection model trained in Amazon SageMaker.

To use a pretrained model, in the `CreateTrainingJob (p. 931)` request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `base_network` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker object detection algorithm, see Amazon SageMaker Object Detection Incremental Training sample notebook. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 594).

**EC2 Instance Recommendation for the Object Detection Algorithm**

For object detection, we support the following GPU instances for training: `ml.p2.xlarge`, `ml.p2.8xlarge`, `ml.p2.16xlarge`, `ml.p3.2xlarge`, `ml.p3.8xlarge` and `ml.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training. However, both CPU (such as C5 and M5) and GPU (such as P2 and P3) instances can be used for the inference. All the supported instance types for inference are itemized on Amazon SageMaker ML Instance Types.

**Object Detection Sample Notebooks**

For a sample notebook that shows how to use the Amazon SageMaker Object Detection algorithm to train and host a model on the COCO dataset using the Single Shot multibox Detector algorithm, see Object Detection using the Image and JSON format. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**How Object Detection Works**

The object detection algorithm identifies and locates all instances of objects in an image from a known collection of object categories. The algorithm takes an image as input and outputs the category that the object belongs to, along with a confidence score that it belongs to the category. The algorithm also predicts the object's location and scale with a rectangular bounding box. Amazon SageMaker Object Detection uses the Single Shot multibox Detector (SSD) algorithm that takes a convolutional neural network (CNN) pretrained for classification task as the base network. SSD uses the output of intermediate layers as features for detection.

Various CNNs such as VGG and ResNet have achieved great performance on the image classification task. Object detection in Amazon SageMaker supports both VGG-16 and ResNet-50 as a base network for SSD. The algorithm can be trained in full training mode or in transfer learning mode. In full training mode, the
The base network is initialized with random weights and then trained on user data. In transfer learning mode, the base network weights are loaded from pretrained models.

The object detection algorithm uses standard data augmentation operations, such as flip, rescale, and jitter, on the fly internally to help avoid overfitting.

**Object Detection Hyperparameters**

In the `CreateTrainingJob` request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters that are used to help estimate the parameters of the model from a training dataset. The following table lists the hyperparameters provided by Amazon SageMaker for training the object detection algorithm. For more information about how object training works, see *How Object Detection Works*.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_classes</code></td>
<td>The number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>num_training_samples</code></td>
<td>The number of training examples in the input dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter will be undefined and distributed training accuracy may be affected.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><code>base_network</code></td>
<td>The base network architecture to use.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 'vgg-16' or 'resnet-50'</td>
</tr>
<tr>
<td></td>
<td>Default value: 'vgg-16'</td>
</tr>
<tr>
<td><code>early_stopping</code></td>
<td>True to use early stopping logic during training. <code>False</code> not to use it.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td><code>early_stopping_min_epochs</code></td>
<td>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when <code>early_stopping = True</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of epochs to wait before ending training if no improvement, as defined by the early_stopping_tolerance hyperparameter, is made in the relevant metric. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>The tolerance value that the relative improvement in validation:mAP, the mean average precision (mAP), is required to exceed to avoid early stopping. If the ratio of the change in the mAP divided by the previous best mAP is smaller than the early_stopping_tolerance value set, early stopping considers that there is no improvement. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>image_shape</td>
<td>The image size for input images. We rescale the input image to a square image with this size. We recommend using 300 and 512 for better performance.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer ≥300</td>
</tr>
<tr>
<td></td>
<td>Default: 300</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of training epochs.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default: 30</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>freeze_layer_pattern</td>
<td>The regular expression (regex) for freezing layers in the base network. For example, if we set `freeze_layer_pattern = &quot;^(conv1_</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default: No layers frozen.</td>
</tr>
<tr>
<td>kv_store</td>
<td>The weight update synchronization mode used for distributed training. The weights can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See the <a href="https://mxnet.apache.org/tutorials/distributed-training.html">Distributed Training MXNet tutorial</a> for details.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is not applicable to single machine training.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: 'dist_sync' or 'dist_async'</td>
</tr>
<tr>
<td></td>
<td>* 'dist_sync': The gradients are synchronized after every batch with all the workers. With 'dist_sync', batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like a single machine with batch size n*b.</td>
</tr>
<tr>
<td></td>
<td>* 'dist_async': Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed.</td>
</tr>
<tr>
<td></td>
<td>Default: -</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| label_width         | The force padding label width used to sync across training and validation data. For example, if one image in the data contains at most 10 objects, and each object's annotation is specified with 5 numbers, [class_id, left, top, width, height], then the label_width should be no smaller than (10*5 + header information length). The header information length is usually 2. We recommend using a slightly larger label_width for the training, such as 60 for this example. Optional  
Valid values: Positive integer large enough to accommodate the largest annotation information length in the data.  
Default: 350 |
| learning_rate       | The initial learning rate. Optional valid values: float in (0, 1]  
Default: 0.001  |
| lr_scheduler_factor | The ratio to reduce learning rate. Used in conjunction with the lr_scheduler_step parameter defined as lr_new = lr_old * lr_scheduler_factor. Optional valid values: float in (0, 1]  
Default: 0.1  |
| lr_scheduler_step   | The epochs at which to reduce the learning rate. The learning rate is reduced by lr_scheduler_factor at epochs listed in a comma-delimited string: "epoch1, epoch2, ...". For example, if the value is set to "10, 20" and the lr_scheduler_factor is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch. Optional  
Valid values: string  
Default: empty string |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. In a single-machine multi-gpu setting, each GPU handles mini_batch_size/num_gpu training samples. For the multi-machine training in dist_sync mode, the actual batch size is mini_batch_size*number of machines. A large mini_batch_size usually leads to faster training, but it may cause out of memory problem. The memory usage is related to mini_batch_size, image_shape, and base_network architecture. For example, on a single p3.2xlarge instance, the largest mini_batch_size without an out of memory error is 32 with the base_network set to &quot;resnet-50&quot; and an image_shape of 300. With the same instance, you can use 64 as the mini_batch_size with the base network vgg-16 and an image_shape of 300.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default: 32</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for sgd. Ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1]</td>
</tr>
<tr>
<td></td>
<td>Default: 0.9</td>
</tr>
<tr>
<td>nms_threshold</td>
<td>The non-maximum suppression threshold.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1]</td>
</tr>
<tr>
<td></td>
<td>Default: 0.45</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer types. For details on optimizer values, see <a href="https://mxnet.apache.org/mxnet/api/index.html">MXNet's API</a>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta']</td>
</tr>
<tr>
<td></td>
<td>Default: 'sgd'</td>
</tr>
<tr>
<td>overlap_threshold</td>
<td>The evaluation overlap threshold.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1]</td>
</tr>
<tr>
<td></td>
<td>Default: 0.5</td>
</tr>
</tbody>
</table>
### Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Optional</th>
<th>Valid values:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_pretrained_model</td>
<td>Indicates whether to use a pre-trained model for training. If set to 1, then the pre-trained model with corresponding architecture is loaded and used for training. Otherwise, the network is trained from scratch.</td>
<td>Optional</td>
<td>0 or 1</td>
<td>1</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The weight decay coefficient for sgd and rmsprop. Ignored for other optimizers.</td>
<td>Optional</td>
<td>float in (0, 1)</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

### Tune an Object Detection Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning](p. 555).

### Metrics Computed by the Object Detection Algorithm

The object detection algorithm reports on a single metric during training: `validation:mAP`. When tuning a model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:mAP</td>
<td>Mean Average Precision (mAP) computed on the validation set.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### Tunable Object Detection Hyperparameters

Tune the Amazon SageMaker object detection model with the following hyperparameters. The hyperparameters that have the greatest impact on the object detection objective metric are: `mini_batch_size`, `learning_rate`, and `optimizer`.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-6, MaxValue: 0.5</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 64</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
<td>Recommended Ranges</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'rmsprop', 'adadelta']</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
</tbody>
</table>

**Object Detection Request and Response Formats**

**Request Format**

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with `image/jpeg` and `image/png` content-types.

**Response Formats**

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

```json
{"prediction":
  [[4.0, 0.8641945540949988, 0.3088374733924866, 0.07030484080314636, 0.7110607028007507, 0.9345266819000244],
   [0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571, 0.9712159633636475],
   [4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057, 0.5967587828636169],
   [8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051, 0.8985623121261597],
   [3.0, 0.42260299175977707, 0.019305512309074402, 0.08386176824569702, 0.39093565940856934, 0.9574796557426453]]}
```

Each row in this .json file contains an array that represents a detected object. Each of these object arrays consists of a list of six numbers. The first number is the predicted class label. The second number is the associated confidence score for the detection. The last four numbers represent the bounding box coordinates [xmin, ymin, xmax, ymax]. These output bounding box corner indices are normalized by the overall image size. Note that this encoding is different than that use by the input .json format. For example, in the first entry of the detection result, 0.3088374733924866 is the left coordinate (x-coordinate of upper-left corner) of the bounding box as a ratio of the overall image width, 0.07030484080314636 is the top coordinate (y-coordinate of upper-left corner) of the bounding box as a ratio of the overall image height, 0.7110607028007507 is the right coordinate (x-coordinate of lower-right corner) of the bounding box as a ratio of the overall image width, and 0.9345266819000244 is the bottom coordinate (y-coordinate of lower-right corner) of the bounding box as a ratio of the overall image height.

To avoid unreliable detection results, you might want to filter out the detection results with low confidence scores. In the object detection sample notebook, we provide scripts to remove the low confidence detections. Scripts are also provided to plot the bounding boxes on the original image.

For batch transform, the response is in JSON format, where the format is identical to the JSON format described above. The detection results of each image is represented as a JSON file. For example:

```json
{"prediction": [[label_id, confidence_score, xmin, ymin, xmax, ymax], [label_id, confidence_score, xmin, ymin, xmax, ymax]]}
```
Principal Component Analysis (PCA) Algorithm

PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called components, which are composites of the original features that are uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

In Amazon SageMaker, PCA operates in two modes, depending on the scenario:

- **regular**: For datasets with sparse data and a moderate number of observations and features.
- **randomized**: For datasets with both a large number of observations and features. This mode uses an approximation algorithm.

PCA uses tabular data.

The rows represent observations you want to embed in a lower dimensional space. The columns represent features that you want to find a reduced approximation for. The algorithm calculates the
covariance matrix (or an approximation thereof in a distributed manner), and then performs the singular value decomposition on this summary to produce the principal components.

**Topics**
- Input/Output Interface for the PCA Algorithm (p. 391)
- EC2 Instance Recommendation for the PCA Algorithm (p. 391)
- PCA Sample Notebooks (p. 391)
- How PCA Works (p. 391)
- PCA Hyperparameters (p. 392)
- PCA Response Formats (p. 393)

**Input/Output Interface for the PCA Algorithm**
For training, PCA expects data provided in the train channel, and optionally supports a dataset passed to the test dataset, which is scored by the final algorithm. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, PCA supports `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Results are returned in either `application/json` or `application/x-recordio-protobuf` format with a vector of "projections."

For more details on training and inference file formats, see the PCA Sample Notebooks (p. 391) and the PCA Response Formats (p. 393).

For more information on input and output file formats, see PCA Response Formats (p. 393) for inference and the PCA Sample Notebooks (p. 391).

**EC2 Instance Recommendation for the PCA Algorithm**
PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

**PCA Sample Notebooks**
For a sample notebook that shows how to use the Amazon SageMaker Principal Component Analysis algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to PCA with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**How PCA Works**
Principal Component Analysis (PCA) is a learning algorithm that reduces the dimensionality (number of features) within a dataset while still retaining as much information as possible.

PCA reduces dimensionality by finding a new set of features called components, which are composites of the original features, but are uncorrelated with one another. The first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

It is an unsupervised dimensionality reduction algorithm. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

Given the input of a matrix with rows $x_1, \ldots, x_n$ each of dimension $1 \times d$, the data is partitioned into mini-batches of rows and distributed among the training nodes (workers). Each worker then computes a
summary of its data. The summaries of the different workers are then unified into a single solution at the end of the computation.

**Modes**

The Amazon SageMaker PCA algorithm uses either of two modes to calculate these summaries, depending on the situation:

* **regular**: for datasets with sparse data and a moderate number of observations and features.
* **randomized**: for datasets with both a large number of observations and features. This mode uses an approximation algorithm.

As the algorithm’s last step, it performs the singular value decomposition on the unified solution, from which the principal components are then derived.

**Mode 1: Regular**

The workers jointly compute both $\sum x_i^T x_i$ and $\sum x_i$.  

**Note**  
Because $x_i$ are 1 * d row vectors, $x_i^T x_i$ is a matrix (not a scalar). Using row vectors within the code allows us to obtain efficient caching.

The covariance matrix is computed as $\sum x_i^T x_i - (1/n)(\sum x_i)^T \sum x_i$, and its top num_components singular vectors form the model.

**Note**  
If subtract_mean is False, we avoid computing and subtracting $\sum x_i$.

Use this algorithm when the dimension d of the vectors is small enough so that $d^3$ can fit in memory.

**Mode 2: Randomized**

When the number of features in the input dataset is large, we use a method to approximate the covariance metric. For every mini-batch $X_i$ of dimension b * d, we randomly initialize a ($\text{num\_components} + \text{extra\_components}$) * b matrix that we multiply by each mini-batch, to create a ($\text{num\_components} + \text{extra\_components}$) * d matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final ($\text{num\_components} + \text{extra\_components}$) * d matrix. The top right num_components singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let $\ell = \text{num\_components} + \text{extra\_components}$. Given a mini-batch $X_i$ of dimension b * d, the worker draws a random matrix $H_i$ of dimension $\ell * b$. Depending on whether the environment uses a GPU or CPU and the dimension size, the matrix is either a random sign matrix where each entry is ±1 or a FJLT (Fast Johnson Lindenstrauss transform; for information, see FJLT Transforms and the follow-up papers). The worker then computes $H_i X_i$ and maintains $B = \sum H_i X_i$. The worker also maintains $h^T$, the sum of columns of $H_1, \ldots, H_T$ (T being the total number of mini-batches), and s, the sum of all input rows. After processing the entire shard of data, the worker sends the server B, h, s, and n (the number of input rows).

Denote the different inputs to the server as $B^i, h^i, s^i, n^i$. The server computes B, h, s, n the sums of the respective inputs. It then computes $C = B - (1/n)h^T s$, and finds its singular value decomposition. The top-right singular vectors and singular values of C are used as the approximate solution to the problem.

**PCA Hyperparameters**

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the...
PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see How PCA Works (p. 391).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>Input dimension.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of rows in a mini-batch.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>num_components</td>
<td>The number of principal components to compute.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>algorithm_mode</td>
<td>Mode for computing the principal components.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: <em>regular</em> or <em>randomized</em></td>
</tr>
<tr>
<td></td>
<td>Default value: <em>regular</em></td>
</tr>
<tr>
<td>extra_components</td>
<td>As the value increases, the solution becomes more accurate but</td>
</tr>
<tr>
<td></td>
<td>the runtime and memory consumption increase linearly. The</td>
</tr>
<tr>
<td></td>
<td>default, -1, means the maximum of 10 and num_components. Valid</td>
</tr>
<tr>
<td></td>
<td>for randomized mode only.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative integer or -1</td>
</tr>
<tr>
<td></td>
<td>Default value: -1</td>
</tr>
<tr>
<td>subtract_mean</td>
<td>Indicates whether the data should be unbiased both during</td>
</tr>
<tr>
<td></td>
<td>training and at inference.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: One of <em>true</em> or <em>false</em></td>
</tr>
<tr>
<td></td>
<td>Default value: <em>true</em></td>
</tr>
</tbody>
</table>

**PCA Response Formats**

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker PCA algorithm.

**JSON Response Format**

Accept—application/json
Random Cut Forest (RCF) Algorithm

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a data set can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

With each data point, RCF associates an anomaly score. Low score values indicate that the data point is considered "normal." High values indicate the presence of an anomaly in the data. The definitions of "low" and "high" depend on the application but common practice suggests that scores beyond three standard deviations from the mean score are considered anomalous.
While there are many applications of anomaly detection algorithms to one-dimensional time series data such as traffic volume analysis or sound volume spike detection, RCF is designed to work with arbitrary-dimensional input. Amazon SageMaker RCF scales well with respect to number of features, data set size, and number of instances.

Topics

• Input/Output Interface for the RCF Algorithm (p. 395)
• Instance Recommendations for the RCF Algorithm (p. 396)
• RCF Sample Notebooks (p. 396)
• How RCF Works (p. 396)
• RCF Hyperparameters (p. 399)
• Tune an RCF Model (p. 399)
• RCF Response Formats (p. 400)

Input/Output Interface for the RCF Algorithm

Amazon SageMaker Random Cut Forest supports the train and test data channels. The optional test channel is used to compute accuracy, precision, recall, and F1-score metrics on labeled data. Train and test data content types can be either application/x-recordio-protobuf or text/csv formats. For the test data, when using text/csv format, the content must be specified as text/csv;label_size=1 where the first column of each row represents the anomaly label: "1" for an anomalous data point and "0" for a normal data point. You can use either File mode or Pipe mode to train RCF models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

Also note that the train channel only supports S3DataDistributionType=ShardedByS3Key and the test channel only supports S3DataDistributionType=FullyReplicated. The S3 distribution type can be specified using the Python SDK as follows:

```python
import sagemaker

# specify Random Cut Forest training job information and hyperparameters
rcf = sagemaker.estimator.Estimator(...)  

# explicitly specify "SharededByS3Key" distribution type
train_data = sagemaker.s3_input(
    s3_data=s3_training_data_location,
    content_type='text/csv;label_size=0',
    distribution='ShardedByS3Key')

# run the training job on input data stored in S3
rcf.fit({'train': train_data})
```

See the Amazon SageMaker Data Types documentation for more information on customizing the S3 data source attributes. Finally, in order to take advantage of multi-instance training the training data must be partitioned into at least as many files as instances.

For inference, RCF supports application/x-recordio-protobuf, text/csv and application/json input data content types. See the Common Data Formats for Built-in Algorithms (p. 228) documentation for more information. RCF inference returns application/x-recordio-protobuf or application/json formatted output. Each record in these output data contains the corresponding anomaly scores for each input data point. See Common Data Formats--Inference for more information.

For more information on input and output file formats, see RCF Response Formats (p. 400) for inference and the RCF Sample Notebooks (p. 396).
Instance Recommendations for the RCF Algorithm

For training, we recommend the ml.m4, ml.c4, and ml.c5 instance families. For inference we recommend using a ml.c5.xl instance type in particular, for maximum performance as well as minimized cost per hour of usage. Although the algorithm could technically run on GPU instance types it does not take advantage of GPU hardware.

RCF Sample Notebooks

For an example of how to train an RCF model and perform inferences with it, see the Introduction to SageMaker Random Cut Forests notebook. For a sample notebook that uses the Amazon SageMaker Random Cut Forest algorithm for anomaly detection, see An Introduction to SageMaker Random Cut Forests. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its Use tab and select Create copy.

How RCF Works

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the “regular” data. Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the “regular” data can often be described with a simple model.

The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree. The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree. The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size. The RCF algorithm is based on the one described in reference [1].

Sample Data Randomly

The first step in the RCF algorithm is to obtain a random sample of the training data. In particular, suppose we want a sample of size $K$ from $N$ total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw $K$ elements from this set. However, frequently the training data is too large to fit all at once, and this approach isn’t feasible. Instead, we use a technique called reservoir sampling.

Reservoir sampling is an algorithm for efficiently drawing random samples from a dataset $S = \{S_1, \ldots, S_K\}$ where the elements in the dataset can only be observed one at a time or in batches. In fact, reservoir sampling works even when $N$ is not known a priori. If only one sample is requested, such as when $K = 1$, the algorithm is like this:

Algorithm: Reservoir Sampling

- Input: dataset or data stream $S = \{S_1, \ldots, S_K\}$
- Initialize the random sample $X = S_1$
- For each observed sample $S_n, n = 2, \ldots, N$:
  - Pick a uniform random number $\xi \in [0, 1]$
This algorithm selects a random sample such that $P(X = S_n) = 1/N$ for all $n = 1, \ldots, N$. When $K > 1$ the algorithm is more complicated. Additionally, a distinction must be made between random sampling that is with and without replacement. RCF performs an augmented reservoir sampling without replacement on the training data based on the algorithms described in [2].

**Train a RCF Model and Produce Inferences**

The next step in RCF is to construct a random cut forest using the random sample of data. First, the sample is partitioned into a number of equal-sized partitions equal to the number of trees in the forest. Then, each partition is sent to an individual tree. The tree recursively organizes its partition into a binary tree by partitioning the data domain into bounding boxes.

This procedure is best illustrated with an example. Suppose a tree is given the following two-dimensional dataset. The corresponding tree is initialized to the root node:

A two-dimensional dataset where the majority of data lies in a cluster (blue) except for one anomalous data point (orange). The tree is initialized with a root node.

The RCF algorithm organizes these data in a tree by first computing a bounding box of the data, selecting a random dimension (giving more weight to dimensions with higher "variance"), and then randomly determining the position of a hyperplane "cut" through that dimension. The two resulting subspaces define their own sub tree. In this example, the cut happens to separate a lone point from the remainder of the sample. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.
A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

Bounding boxes are then computed for the left and right halves of the data and the process is repeated until every leaf of the tree represents a single data point from the sample. Note that if the lone point is sufficiently far away then it is more likely that a random cut would result in point isolation. This observation provides the intuition that tree depth is, loosely speaking, inversely proportional to the anomaly score.

When performing inference using a trained RCF model the final anomaly score is reported as the average across scores reported by each tree. Note that it is often the case that the new data point does not already reside in the tree. To determine the score associated with the new point the data point is inserted into the given tree and the tree is efficiently (and temporarily) reassembled in a manner equivalent to the training process described above. That is, the resulting tree is as if the input data point were a member of the sample used to construct the tree in the first place. The reported score is inversely proportional to the depth of the input point within the tree.

**Choose Hyperparameters**

The primary hyperparameters used to tune the RCF model are `num_trees` and `num_samples_per_tree`. Increasing `num_trees` has the effect of reducing the noise observed in anomaly scores since the final score is the average of the scores reported by each tree. While the optimal value is application-dependent we recommend using 100 trees to begin with as a balance between score noise and model complexity. Note that inference time is proportional to the number of trees. Although training time is also affected it is dominated by the reservoir sampling algorithm described above.

The parameter `num_samples_per_tree` is related to the expected density of anomalies in the dataset. In particular, `num_samples_per_tree` should be chosen such that $1/\text{num samples per tree}$ approximates the ratio of anomalous data to normal data. For example, if 256 samples are used in each tree then we expect our data to contain anomalies 1/256 or approximately 0.4% of the time. Again, an optimal value for this hyperparameter is dependent on the application.

**References**


**RCF Hyperparameters**

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker RCF algorithm. For more information, including recommendations on how to choose hyperparameters, see How RCF Works (p. 396).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The number of features in the data set. (If you are using the client libraries through a notebook, this value is calculated for you and need not be specified.) Required (When the job is run through the console.) Valid values: Positive integer (min: 1, max: 10000)</td>
</tr>
</tbody>
</table>
| eval_metrics           | A list of metrics used to score a labeled test data set. The following metrics can be selected for output:  
  • accuracy - returns fraction of correct predictions.  
  • precision_recall_fscore - returns the positive and negative precision, recall, and F1-scores. Optional Valid values: a list with possible values taken from accuracy or precision_recall_fscore. Default value: Both accuracy, precision_recall_fscore are calculated. |
| num_samples_per_tree   | Number of random samples given to each tree from the training data set. Optional Valid values: Positive integer (min: 1, max: 2048) Default value: 256 |
| num_trees              | Number of trees in the forest. Optional Valid values: Positive integer (min: 50, max: 1000) Default value: 100 |

**Tune an RCF Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable
hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker RCF algorithm is an unsupervised anomaly-detection algorithm that requires a labeled test dataset for hyperparameter optimization. It calculates anomaly scores for test datapoints and then labels the datapoints as anomalous if their scores are beyond three standard deviations from the mean score. This is known as the three-sigma limit heuristic. The F1 score is emitted based on the difference between calculated labels and actual labels. The hyperparameter tuning job finds the model that maximizes that score. The success of hyperparameter optimization depends on the applicability of the three-sigma limit heuristic to the test dataset.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

### Metrics Computed by the RCF Algorithm

The RCF algorithm computes the following metric during training. When tuning the model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:f1</td>
<td>F1 score on the test dataset, based on the difference between calculated labels and actual labels.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### Tunable RCF Hyperparameters

You can tune a RCF model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_samples_per_tree</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue:2048</td>
</tr>
<tr>
<td>num_trees</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 50, MaxValue:1000</td>
</tr>
</tbody>
</table>

### RCF Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. Note that Amazon SageMaker Random Cut Forest supports both dense and sparse JSON and RecordIO formats. This topic contains a list of the available output formats for the Amazon SageMaker RCF algorithm.

#### JSON Response Format

ACCEPT: application/json.

```json
{
    "scores": [400]
}
```
Use Built-in Algorithms

JSONLINES Response Format
ACCEPT: application/jsonlines.

 RECORDIO Response Format
ACCEPT: application/x-recordio-protobuf.
Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm provides a fine-grained, pixel-level approach to developing computer vision applications. It tags every pixel in an image with a class label from a predefined set of classes. Tagging is fundamental for understanding scenes, which is critical to an increasing number of computer vision applications, such as self-driving vehicles, medical imaging diagnostics, and robot sensing.

For comparison, the Amazon SageMaker Image Classification Algorithm (p. 271) is a supervised learning algorithm that analyzes only whole images, classifying them into one of multiple output categories. The Object Detection Algorithm (p. 365) is a supervised learning algorithm that detects and classifies all instances of an object in an image. It indicates the location and scale of each object in the image with a rectangular bounding box.

Because the semantic segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects contained in the image. The segmentation output is represented as an RGB or grayscale image, called a segmentation mask. A segmentation mask is an RGB (or grayscale) image with the same shape as the input image.

Amazon SageMaker semantic segmentation algorithm is built using the MXNet Gluon framework and the Gluon CV toolkit provides you with a choice of three built-in algorithms to train a deep neural network.
You can use the Fully-Convolutional Network (FCN) algorithm, Pyramid Scene Parsing (PSP) algorithm, or DeepLabV3.

Each of the three algorithms has two distinct components:

- The **backbone** (or **encoder**)—A network that produces reliable activation maps of features.
- The **decoder**—A network that constructs the segmentation mask from the encoded activation maps.

You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: ResNet50 or ResNet101. These backbones include pretrained artifacts that were originally trained on the ImageNet classification task. You can fine-tune these backbones for segmentation using your own data. Or, you can initialize and train these networks from scratch using only your own data. The decoders are never pretrained.

To deploy the trained model for inference, use the Amazon SageMaker hosting service. During inference, you can request the segmentation mask either as a PNG image or as a set of probabilities for each class for each pixel. You can use these masks as part of a larger pipeline that includes additional downstream image processing or other applications.

**Topics**

- Semantic Segmentation Sample Notebooks (p. 403)
- Input/Output Interface for the Semantic Segmentation Algorithm (p. 403)
- EC2 Instance Recommendation for the Semantic Segmentation Algorithm (p. 406)
- Semantic Segmentation Hyperparameters (p. 406)

**Semantic Segmentation Sample Notebooks**

For a sample Jupyter notebook that uses the Amazon SageMaker semantic segmentation algorithm to train a model and deploy it to perform inferences, see the Semantic Segmentation Example. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201).

To see a list of all of the Amazon SageMaker samples, create and open a notebook instance, and choose the SageMaker Examples tab. The example semantic segmentation notebooks are located under Introduction to Amazon algorithms. To open a notebook, choose its Use tab, and choose Create copy.

**Input/Output Interface for the Semantic Segmentation Algorithm**

Amazon SageMaker semantic segmentation expects the customer’s training dataset to be on Amazon Simple Storage Service (Amazon S3). Once trained, it produces the resulting model artifacts on Amazon S3. The input interface format for the Amazon SageMaker semantic segmentation is similar to that of most standardized semantic segmentation benchmarking datasets. The dataset in Amazon S3 is expected to be presented in two channels, one for train and one for validation using four directories, two for images and two for annotations. Annotations are expected to be uncompressed PNG images. The dataset might also have a label map that describes how the annotation mappings are established. If not, the algorithm uses a default. It also supports the augmented manifest image format (application/x-image) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an image/jpeg content type.

**How Training Works**

The training data is split into four directories: train, train_annotation, validation, and validation_annotation. There is a channel for each of these directories. The dataset also expected to have one label_map.json file per channel for train_annotation and validation_annotation respectively. If you don't provide these JSON files, Amazon SageMaker provides the default set label map.
The dataset specifying these files should look similar to the following example:

```plaintext
s3://bucket_name
  |- train
  |   |- 0000.jpg
  |   |- coffee.jpg
  |- validation
  |   |- 00a0.jpg
  |   |- banana.jpg
  |- train_annotation
  |   |- 0000.png
  |   |- coffee.png
  |- validation_annotation
  |   |- 00a0.png
  |   |- banana.png
  |- label_map
    |- train_label_map.json
    |- validation_label_map.json
```

Every JPG image in the train and validation directories has a corresponding PNG label image with the same name in the `train_annotation` and `validation_annotation` directories. This naming convention helps the algorithm to associate a label with its corresponding image during training. The `train`, `train_annotation`, `validation`, and `validation_annotation` channels are mandatory. The annotations are single-channel PNG images. The format works as long as the metadata (modes) in the image helps the algorithm read the annotation images into a single-channel 8-bit unsigned integer. For more information on our support for modes, see the Python Image Library documentation. We recommend using the 8-bit pixel, true color P mode.

The image that is encoded is a simple 8-bit integer when using modes. To get from this mapping to a map of a label, the algorithm uses one mapping file per channel, called the label map. The label map is used to map the values in the image with actual label indices. In the default label map, which is provided by default if you don’t provide one, the pixel value in an annotation matrix (image) directly index the label. These images can be grayscale PNG files or 8-bit indexed PNG files. The label map file for the unscaled default case is the following:

```json
{
    "scale": "1"
}
```

To provide some contrast for viewing, some annotation software scales the label images by a constant amount. To support this, the Amazon SageMaker semantic segmentation algorithm provides a rescaling option to scale down the values to actual label values. When scaling down doesn’t convert the value to an appropriate integer, the algorithm defaults to the greatest integer less than or equal to the scale value. The following code shows how to set the scale value to rescale the label values:

```json
{
    "scale": "3"
}
```

The following example shows how this "scale" value is used to rescale the encoded_label values of the input annotation image when they are mapped to the mapped_label values to be used in training. The label values in the input annotation image are 0, 3, 6, with scale 3, so they are mapped to 0, 1, 2 for training:
encoded_label = [0, 3, 6]
mapped_label = [0, 1, 2]

In some cases, you might need to specify a particular color mapping for each class. Use the map option in the label mapping as shown in the following example of a label_map file:

```json
{
  "map": {
    "0": 5,
    "1": 0,
    "2": 2
  }
}
```

This label mapping for this example is:

encoded_label = [0, 5, 2]
mapped_label = [1, 0, 2]

With label mappings, you can use different annotation systems and annotation software to obtain data without a lot of preprocessing. You can provide one label map per channel. The files for a label map in the label_map channel must follow the naming conventions for the four directory structure. If you don't provide a label map, the algorithm assumes a scale of 1 (the default).

**Training with the Augmented Manifest Format**

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. The augmented manifest file contains data objects and should be in JSON Lines format, as described in the CreateTrainingJob (p. 931) request API. Each line in the manifest is an entry containing the Amazon S3 URI for the image and the URI for the annotation image.

Each JSON object in the manifest file must contain a source-ref key. The source-ref key should contain the value of the Amazon S3 URI to the image. The labels are provided under the AttributeNames parameter value as specified in the CreateTrainingJob (p. 931) request. It can also contain additional metadata under the metadata tag, but these are ignored by the algorithm. In the example below, the AttributeNames are contained in the list of image and annotation references ["source-ref", "city-streets-ref"]. These names must have -ref appended to them. When using the Semantic Segmentation algorithm with Augmented Manifest, the value of the RecordWrapperType parameter must be "RecordIO" and value of the ContentType parameter must be application/x-recordio.

```json
{"source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-streets-metadata":{"job-name": "label-city-streets", }}
```

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

**Incremental Training**

You can also seed the training of a new model with a model that you trained previously using Amazon SageMaker. This incremental training saves training time when you want to train a new model with the same or similar data. Currently, incremental training is supported only for models trained with the built-in Amazon SageMaker Semantic Segmentation.

To use your own pre-trained model, specify the ChannelName as "model" in the InputDataConfig for the CreateTrainingJob (p. 931) request. Set the ContentType for the model channel to application/x-sagemaker-model. The backbone, algorithm, crop_size, and num_classes input parameters that define the network architecture must be consistently specified in the input
hyperparameters of the new model and the pre-trained model that you upload to the model channel. For the pretrained model file, you can use the compressed (.tar.gz) artifacts from Amazon SageMaker outputs. You can use either RecordIO or Image formats for input data. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 594).

Produce Inferences

To query a trained model that is deployed to an endpoint, you need to provide an image and an AcceptType that denotes the type of output required. The endpoint takes JPEG images with an image/jpeg content type. If you request an AcceptType of image/png, the algorithm outputs a PNG file with a segmentation mask in the same format as the labels themselves. If you request an accept type of application/x-recordio-protobuf, the algorithm returns class probabilities encoded in recordio-protobuf format. The latter format outputs a 3D tensor where the third dimension is the same size as the number of classes. This component denotes the probability of each class label for each pixel.

EC2 Instance Recommendation for the Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes. The algorithm can be trained using P2/P3 EC2 Amazon Elastic Compute Cloud (Amazon EC2) instances in single machine configurations. It supports the following GPU instances for training:

- ml.p2.xlarge
- ml.p2.8xlarge
- ml.p2.16xlarge
- ml.p3.2xlarge
- ml.p3.8xlarge
- ml.p3.16xlarge

For inference, you can use either CPU instances (such as c5 and m5) and GPU instances (such as p2 and p3) or both. For information about the instance types that provide varying combinations of CPU, GPU, memory, and networking capacity for inference, see Amazon SageMaker ML Instance Types.

Semantic Segmentation Hyperparameters

The following tables list the hyperparameters supported by the Amazon SageMaker semantic segmentation algorithm for network architecture, data inputs, and training. You specify Semantic Segmentation for training in the AlgorithmName of the CreateTrainingJob (p. 931) request.

Network Architecture Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backbone</td>
<td>The backbone to use for the algorithm's encoder component.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: resnet-50, resnet-101</td>
</tr>
<tr>
<td></td>
<td>Default value: resnet-50</td>
</tr>
<tr>
<td>use_pretrained_model</td>
<td>Whether a pretrained model is to be used for the backbone.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True, False</td>
</tr>
</tbody>
</table>
## Use Built-in Algorithms

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default value: True</td>
<td></td>
</tr>
</tbody>
</table>

### algorithm

The algorithm to use for semantic segmentation.

**Optional**

Valid values:
- **fcn**: Fully-Convolutional Network (FCN) algorithm
- **psp**: Pyramid Scene Parsing (PSP) algorithm
- **deeplab**: DeepLab V3 algorithm

Default value: fcn

### Data Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| num_classes | The number of classes to segment. **Required**

Valid values: $2 \leq$ positive integer $\leq 254$

| num_training_samples | The number of samples in the training data. The algorithm uses this value to set up the learning rate scheduler. **Required**

Valid values: positive integer

| crop_size | The image size for input during training. We randomly rescale the input image while preserving the aspect ratio and then take a random square crop with side length crop_size. **Optional**

Valid values: positive integer $> 16$

Default value: 480

### Training Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| early_stopping | Whether to use early stopping logic during training. **Optional**

Valid values: True, False

Default value: False

<p>| early_stopping_min_epochs | The minimum number of epochs that must be run. |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| early_stopping_patience        | The number of epochs that meet the tolerance for lower performance before the algorithm enforces an early stop.  
Optional                                                                 |
|                               | Valid values: integer                                                       |
|                               | Default value: 5                                                            |
| early_stopping_tolerance       | If the relative improvement of the score of the training job, the mIOU, is smaller than this value, early stopping considers the epoch as not improved. This is used only when early_stopping = True.  
Optional                                                                 |
|                               | Valid values: 0 ≤ float ≤ 1                                                  |
|                               | Default value: 0.0                                                          |
| epochs                         | The number of epochs with which to train.                                   
Optional                                                                 |
|                               | Valid values: positive integer                                              |
|                               | Default value: 30                                                           |
| gamma1                         | The decay factor for the moving average of the squared gradient for rmsprop.  
Used only for rmsprop.                                                   
Optional                                                                 |
|                               | Valid values: 0 ≤ float ≤ 1                                                  |
|                               | Default value: 0.9                                                          |
| gamma2                         | The momentum factor for rmsprop.                                            
Optional                                                                 |
|                               | Valid values: 0 ≤ float ≤ 1                                                  |
|                               | Default value: 0.9                                                          |
| learning_rate                  | The initial learning rate.                                                  
Optional                                                                 |
<p>|                               | Valid values: 0 &lt; float ≤ 1                                                  |
|                               | Default value: 0.001                                                        |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr_scheduler</td>
<td>The shape of the learning rate schedule that controls its decrease over time.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• <strong>step</strong>: A stepwise decay, where the learning rate is reduced by a factor at certain intervals.</td>
</tr>
<tr>
<td></td>
<td>• <strong>poly</strong>: A smooth decay using a polynomial function.</td>
</tr>
<tr>
<td></td>
<td>• <strong>cosine</strong>: A smooth decay using a cosine function.</td>
</tr>
<tr>
<td></td>
<td>Default value: <strong>poly</strong></td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. Using a large mini_batch_size usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 4</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for the sgd optimizer. When you use other optimizers, the semantic segmentation algorithm ignores this parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 &lt; \text{float} \leq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>optimizer</td>
<td>The type of optimizer. For more information about an optimizer, choose the appropriate link:</td>
</tr>
<tr>
<td></td>
<td>• <strong>adam</strong>: Adaptive momentum estimation</td>
</tr>
<tr>
<td></td>
<td>• <strong>adagrad</strong>: Adaptive gradient descent</td>
</tr>
<tr>
<td></td>
<td>• <strong>nag</strong>: Nesterov accelerated gradient</td>
</tr>
<tr>
<td></td>
<td>• <strong>rmsprop</strong>: Root mean square propagation</td>
</tr>
<tr>
<td></td>
<td>• <strong>sgd</strong>: Stochastic gradient descent</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: adam, adagrad, nag, rmsprop, sgd</td>
</tr>
<tr>
<td></td>
<td>Default value: sgd</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| validation_mini_batch_size | The batch size for validation. A large mini_batch_size usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture.  
  - To score the validation on the entire image without cropping the images, set this parameter to 1. Use this option if you want to measure performance on the entire image as a whole.  
    **Note**  
    Setting the validation_mini_batch_size parameter to 1 causes the algorithm to create a new network model for every image. This might slow validation and training.  
  - To crop images to the size specified in the crop_size parameter, even during evaluation, set this parameter to a value greater than 1. | Optional      | positive integer                                                             | 4              |
| weight_decay         | The weight decay coefficient for the sgd optimizer. When you use other optimizers, the algorithm ignores this parameter.                                                                                       | Optional      | 0 < float < 1                                                                | 0.0001         |

**Sequence-to-Sequence Algorithm**

Amazon SageMaker Sequence to Sequence is a supervised learning algorithm where the input is a sequence of tokens (for example, text, audio) and the output generated is another sequence of tokens. Example applications include: machine translation (input a sentence from one language and predict what that sentence would be in another language), text summarization (input a longer string of words and predict a shorter string of words that is a summary), speech-to-text (audio clips converted into output sentences in tokens). Recently, problems in this domain have been successfully modeled with deep neural networks that show a significant performance boost over previous methodologies. Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

**Topics**

- [Input/Output Interface for the Sequence-to-Sequence Algorithm](#)
- [EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm](#)
- [Sequence-to-Sequence Sample Notebooks](#)
- [How Sequence-to-Sequence Works](#)
- [Sequence-to-Sequence Hyperparameters](#)
- [Tune a Sequence-to-Sequence Model](#)
Input/Output Interface for the Sequence-to-Sequence Algorithm

Training

Amazon SageMaker seq2seq expects data in RecordIO-Protobuf format. However, the tokens are expected as integers, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in the seq2seq example notebook. In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

- **train**: It should contain the training data (for example, the `train.rec` file generated by the preprocessing script).
- **validation**: It should contain the validation data (for example, the `val.rec` file generated by the preprocessing script).
- **vocab**: It should contain two vocabulary files (`vocab.src.json` and `vocab.trg.json`)

If the algorithm doesn't find data in any of these three channels, training results in an error.

Inference

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the `application/json` format. Otherwise, use the `recordio-protobuf` format to work with the integer encoded data. Both mode supports batching of input data. `application/json` format also allows you to visualize the attention matrix.

- **application/json**: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be `application/json`. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:

  ```json
  configuration: {attention_matrix: true}: Returns the attention matrix for the particular input sequence.
  ```

- **application/x-recordio-protobuf**: Expects the input in `recordio-protobuf` format and returns the output in `recordio-protobuf` format. Both content and accept types should be `application/x-recordio-protobuf`. For this format, the source sequences must be converted into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For batch transform, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be `application/jsonlines`. The format for input is as follows:

```json
content-type: application/jsonlines
{"source": "source_sequence_0"}
{"source": "source_sequence_1"}
```

The format for response is as follows:

```json
accept: application/jsonlines
```
For additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference, see the Sequence-to-Sequence Sample Notebooks (p. 412).

### EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm

Currently Amazon SageMaker seq2seq is only supported on GPU instance types and is only set up to train on a single machine. But it does also offer support for multiple GPUs.

#### Sequence-to-Sequence Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Sequence to Sequence algorithm to train a English-German translation model, see Machine Translation English-German Example Using SageMaker Seq2Seq. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

#### How Sequence-to-Sequence Works

Typically, a neural network for sequence-to-sequence modeling consists of a few layers, including:

- **An embedding layer.** In this layer, the input matrix, which is input tokens encoded in a sparse way (for example, one-hot encoded) are mapped to a dense feature layer. This is required because a high-dimensional feature vector is more capable of encoding information regarding a particular token (word for text corpora) than a simple one-hot-encoded vector. It is also a standard practice to initialize this embedding layer with a pre-trained word vector like FastText or Glove or to initialize it randomly and learn the parameters during training.

- **An encoder layer.** After the input tokens are mapped into a high-dimensional feature space, the sequence is passed through an encoder layer to compress all the information from the input embedding layer (of the entire sequence) into a fixed-length feature vector. Typically, an encoder is made of RNN-type networks like long short-term memory (LSTM) or gated recurrent units (GRU). (Colah's blog explains LSTM in a great detail.)

- **A decoder layer.** The decoder layer takes this encoded feature vector and produces the output sequence of tokens. This layer is also usually built with RNN architectures (LSTM and GRU).

The whole model is trained jointly to maximize the probability of the target sequence given the source sequence. This model was first introduced by Sutskever et al. in 2014.

**Attention mechanism.** The disadvantage of an encoder-decoder framework is that model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain. To tackle this problem, in 2015, Bahdanau et al. proposed the attention mechanism. In an attention mechanism, the decoder tries to find the location in the encoder sequence where the most important information could be located and uses that information and previously decoded words to predict the next token in the sequence.

For more in details, see the whitepaper Effective Approaches to Attention-based Neural Machine Translation by Luong, et al. that explains and simplifies calculations for various attention mechanisms. Additionally, the whitepaper Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation by Wu, et al. describes Google's architecture for machine translation, which uses skip connections between encoder and decoder layers.
## Sequence-to-Sequence Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_size</td>
<td>Mini batch size for gradient descent.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 64</td>
</tr>
<tr>
<td>beam_size</td>
<td>Length of the beam for beam search. Used during training for computing bleu and used during inference.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>bleu_sample_size</td>
<td>Number of instances to pick from validation dataset to decode and compute bleu score during training. Set to -1 to use full validation set (if bleu is chosen as optimized_metric).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>bucket_width</td>
<td>Returns (source,target) buckets up to (max_seq_len_source, max_seq_len_target). The longer side of the data uses steps of bucket_width while the shorter side uses steps scaled down by the average target/source length ratio. If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of max_len.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>bucketing_enabled</td>
<td>Set to false to disable bucketing, unroll to maximum length.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>checkpoint_frequency_num_batches</td>
<td>Checkpoint and evaluate every x batches.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>checkpoint_threshold</td>
<td>Maximum number of checkpoints model is allowed to not improve in optimized_metric on validation dataset before training is stopped.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Default value: 3</td>
<td></td>
</tr>
<tr>
<td>clip_gradient</td>
<td>Clip absolute gradient values greater than this. Set to negative to disable.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: float</td>
</tr>
<tr>
<td>Default value: 1</td>
<td></td>
</tr>
<tr>
<td>cnn_activation_type</td>
<td>The cnn activation type to be used.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: String. One of glu, relu, softrelu, sigmoid, or tanh.</td>
</tr>
<tr>
<td>Default value: glu</td>
<td></td>
</tr>
<tr>
<td>cnn_hidden_dropout</td>
<td>Dropout probability for dropout between convolutional layers.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
<tr>
<td>cnn_kernel_width_decoder</td>
<td>Kernel width for the cnn decoder.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Default value: 5</td>
<td></td>
</tr>
<tr>
<td>cnn_kernel_width_encoder</td>
<td>Kernel width for the cnn encoder.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Default value: 3</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cnn_num_hidden</td>
<td>Number of cnn hidden units for encoder and decoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td>decoder_type</td>
<td>Decoder type.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td>embed_dropout_source</td>
<td>Dropout probability for source side embeddings.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>embed_dropout_target</td>
<td>Dropout probability for target side embeddings.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>encoder_type</td>
<td>Encoder type. The rnn architecture is based on attention mechanism by Bahdanau et al. and cnn architecture is based on Gehring et al.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td>fixed_rate_lr_half_life</td>
<td>Half life for learning rate in terms of number of checkpoints for fixed_rate_* schedulers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Initial learning rate.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0003</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>loss_type</td>
<td>Loss function for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. cross-entropy</td>
</tr>
<tr>
<td></td>
<td>Default value: cross-entropy</td>
</tr>
<tr>
<td>lr_scheduler_type</td>
<td>Learning rate scheduler type. plateau_reduce means reduce the learning rate whenever optimized_metric on validation_accuracy plateaus. inv_t is inverse time decay. learning_rate/(1+decay_rate*t)</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of plateau_reduce, fixed_rate_inv_t, or fixed_rate_inv_sqrt_t.</td>
</tr>
<tr>
<td></td>
<td>Default value: plateau_reduce</td>
</tr>
<tr>
<td>max_num_batches</td>
<td>Maximum number of updates/batches to process. -1 for infinite.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -1</td>
</tr>
<tr>
<td>max_num_epochs</td>
<td>Maximum number of epochs to pass through training data before fitting is stopped. Training continues until this number of epochs even if validation accuracy is not improving if this parameter is passed. Ignored if not passed.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer and less than or equal to max_num_epochs.</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td>max_seq_len_source</td>
<td>Maximum length for the source sequence length. Sequences longer than this length are truncated to this length.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>max_seq_len_target</td>
<td>Maximum length for the target sequence length. Sequences longer than this length are truncated to this length.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>min_num_epochs</strong></td>
<td>Minimum number of epochs the training must run before it is stopped via early_stopping conditions.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>momentum</strong></td>
<td>Momentum constant used for sgd. Don't pass this parameter if you are using adam or rmsprop.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td><strong>num_embed_source</strong></td>
<td>Embedding size for source tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><strong>num_embed_target</strong></td>
<td>Embedding size for target tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><strong>num_layers_decoder</strong></td>
<td>Number of layers for Decoder rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td><strong>num_layers_encoder</strong></td>
<td>Number of layers for Encoder rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td><strong>optimized_metric</strong></td>
<td>Metrics to optimize with early stopping.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of perplexity, accuracy, or bleu.</td>
</tr>
<tr>
<td></td>
<td>Default value: perplexity</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer_type</td>
<td>Optimizer to choose from.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of adam, sgd, or rmsprop.</td>
</tr>
<tr>
<td></td>
<td>Default value: adam</td>
</tr>
<tr>
<td>plateau_reduce_lr_factor</td>
<td>Factor to multiply learning rate with (for plateau_reduce).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>plateau_reduce_lr_threshold</td>
<td>For plateau_reduce scheduler, multiply learning rate with reduce factor if optimized_metric didn't improve for this many checkpoints.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>rnn_attention_in_upper_layers</td>
<td>Pass the attention to upper layers of rnn, like Google NMT paper. Only applicable if more than one layer is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: boolean (true or false)</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>rnn_attention_num_hidden</td>
<td>Number of hidden units for attention layers. defaults to rnn_num_hidden.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn_num_hidden</td>
</tr>
<tr>
<td>rnn_attention_type</td>
<td>Attention model for encoders. mlp refers to concat and bilinear refers to general from the Luong et al. paper.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of dot, fixed, mlp, or bilinear.</td>
</tr>
<tr>
<td></td>
<td>Default value: mlp</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>rnn_cell_type</code></td>
<td>Specific type of \textit{rnn} architecture. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: String. Either \texttt{lstm} or \texttt{gru}. \hspace{1cm} Default value: \texttt{lstm}</td>
</tr>
<tr>
<td><code>rnn_decoder_state_init</code></td>
<td>How to initialize \textit{rnn} decoder states from encoders. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: String. One of \texttt{last}, \texttt{avg}, or \texttt{zero}. \hspace{1cm} Default value: \texttt{last}</td>
</tr>
<tr>
<td><code>rnn_first_residual_layer</code></td>
<td>First \textit{rnn} layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: positive integer \hspace{1cm} Default value: 2</td>
</tr>
<tr>
<td><code>rnn_num_hidden</code></td>
<td>The number of \textit{rnn} hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional Long Term Short Term Memory (LSTM) by default. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: positive even integer \hspace{1cm} Default value: 1024</td>
</tr>
<tr>
<td><code>rnn_residual_connections</code></td>
<td>Add residual connection to stacked \textit{rnn}. Number of layers should be more than 1. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: boolean (\texttt{true} or \texttt{false}) \hspace{1cm} Default value: \texttt{false}</td>
</tr>
<tr>
<td><code>rnn_decoder_hidden_dropout</code></td>
<td>Dropout probability for hidden state that combines the context with the \textit{rnn} hidden state in the decoder. \hspace{1cm} \textbf{Optional} \hspace{1cm} Valid values: Float. Range in [0,1]. \hspace{1cm} Default value: 0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>training_metric</td>
<td>Metrics to track on training on validation data.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <code>perplexity</code> or <code>accuracy</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>perplexity</code></td>
</tr>
<tr>
<td>weight_decay</td>
<td>Weight decay constant.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>weight_init_scale</td>
<td>Weight initialization scale (for uniform and xavier initialization).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 2.34</td>
</tr>
<tr>
<td>weight_init_type</td>
<td>Type of weight initialization.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <code>uniform</code> or <code>xavier</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>xavier</code></td>
</tr>
<tr>
<td>xavier_factor_type</td>
<td>Xavier factor type.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <code>in</code>, <code>out</code>, or <code>avg</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>in</code></td>
</tr>
</tbody>
</table>

**Tune a Sequence-to-Sequence Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

**Metrics Computed by the Sequence-to-Sequence Algorithm**

The sequence to sequence algorithm reports three metrics that are computed during training. Choose one of them as an objective to optimize when tuning the hyperparameter values.
<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:accuracy</td>
<td>Accuracy computed on the validation dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:bleu</td>
<td>Bleu score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the <code>bleu_sample_size</code> parameter to specify the subsample.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:perplexity</td>
<td>Perplexity, is a loss function computed on the validation dataset. Perplexity measures the cross-entropy between an empirical sample and the distribution predicted by a model and so provides a measure of how well a model predicts the sample values, Models that are good at predicting a sample have a low perplexity.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

**Tunable Sequence-to-Sequence Hyperparameters**

You can tune the following hyperparameters for the Amazon SageMaker Sequence to Sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective metrics are: `batch_size`, `optimizer_type`, `learning_rate`, `num_layers_encoder`, and `num_layers_decoder`.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_layers_encoder</td>
<td>IntegerParameterRange</td>
<td>[1-10]</td>
</tr>
<tr>
<td>num_layers_decoder</td>
<td>IntegerParameterRange</td>
<td>[1-10]</td>
</tr>
<tr>
<td>batch_size</td>
<td>CategoricalParameterRange</td>
<td>[16,32,64,128,256,512,1024,2048]</td>
</tr>
<tr>
<td>optimizer_type</td>
<td>CategoricalParameterRange</td>
<td>['adam', 'sgd', 'rmsprop']</td>
</tr>
<tr>
<td>weight_init_type</td>
<td>CategoricalParameterRange</td>
<td>['xavier', 'uniform']</td>
</tr>
<tr>
<td>weight_init_scale</td>
<td>ContinuousParameterRange</td>
<td>For the xavier type: MinValue: 2.0, MaxValue: 3.0. For the uniform type: MinValue: -1.0, MaxValue: 1.0</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.00005, MaxValue: 0.2</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.1</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.5, MaxValue: 0.9</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1.0, MaxValue: 5.0</td>
</tr>
<tr>
<td>rnn_num_hidden</td>
<td>CategoricalParameterRange</td>
<td>Applicable only to recurrent neural</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
<td>Recommended Ranges</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>networks (RNNs)</td>
<td></td>
<td>[128, 256, 512, 1024, 2048]</td>
</tr>
<tr>
<td>cnn_num_hidden</td>
<td>CategoricalParameterRange</td>
<td>Applicable only to convolutional neural networks (CNNs). [128, 256, 512, 1024, 2048]</td>
</tr>
<tr>
<td>num_embed_source</td>
<td>IntegerParameterRange</td>
<td>[256-512]</td>
</tr>
<tr>
<td>num_embed_target</td>
<td>IntegerParameterRange</td>
<td>[256-512]</td>
</tr>
<tr>
<td>embed_dropout_source</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>embed_dropout_target</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>rnn_decoder_hidden</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>cnn_hidden_dropout</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>lr_scheduler_type</td>
<td>CategoricalParameterRange</td>
<td>['plateau_reduce', 'fixed_rate_inv_t', 'fixed_rate_inv_sqrt_t']</td>
</tr>
<tr>
<td>plateau_reduce_lr_factor</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.1, MaxValue: 0.5</td>
</tr>
<tr>
<td>plateau_reduce_lr_threshold</td>
<td>IntegerParameterRange</td>
<td>[1-5]</td>
</tr>
<tr>
<td>fixed_rate_lr_half_life</td>
<td>IntegerParameterRange</td>
<td>[10-30]</td>
</tr>
</tbody>
</table>

**XGBoost Algorithm**

The XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and because of the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

This current release of the XGBoost algorithm makes upgrades from the open source XGBoost code base easy to install and use in Amazon SageMaker. Customers can use this release of the XGBoost algorithm either as an Amazon SageMaker built-in algorithm, as with the previous 0.72-based version, or as a framework to run training scripts in their local environments as they would typically do, for example, with a TensorFlow deep learning framework. This implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics than the original 0.72-based version. It also provides an XGBoost estimator that executes a training script in a managed XGBoost environment. The current release of Amazon SageMaker XGBoost is based on version 0.90 and will be upgradeable to future releases. The previous implementation XGBoost Release 0.72 (p. 434) is still available to customers if they need to postpone migrating to the current version. But this previous implementation will remain tied to the 0.72 release of XGBoost.
How to Use Amazon SageMaker XGBoost

The XGBoost algorithm can be used as a built-in algorithm or as a framework such as TensorFlow. Using XGBoost as a framework provides more flexibility than using it as a built-in algorithm as it enables more advanced scenarios that allow pre-processing and post-processing scripts to be incorporated into your training script. Using XGBoost as a built-in Amazon SageMaker algorithm is how you had to use the original XGBoost Release 0.72 version and nothing changes here except the version of XGBoost that you use.

• Use XGBoost as a framework

Use XGBoost as a framework to run scripts that can incorporate additional data processing into your training jobs. This way of using XGBoost should be familiar to users who have worked with the open source XGBoost and other Amazon SageMaker frameworks such as Scikit-learn. You use the Amazon SageMaker Python SDK as you would for other frameworks such as TensorFlow. One change from other Amazon SageMaker frameworks is that the framework_version field of the estimator for XGBoost is mandatory and is not set by default. Note that the first part of the version refers to the upstream module version (aka, 0.90), while the second part refers to the Amazon SageMaker version for the container. An error is generated if the framework_version is not set.

```python
import sagemaker.xgboost
estimator = XGBoost(entry_point = 'myscript.py',
                     source_dir, model_dir, train_instance_type,
                     train_instance_count, hyperparameters, role, base_job_name,
                     framework_version = '0.90-1',
                     py_version)
estimator.fit({'train':'s3://my-bucket/training',
                        'validation':'s3://my-bucket/validation'})
```

The AWS SDK for Python (Boto 3) and the CLI also require this field.

• Use XGBoost as a built-in algorithm

Use XGBoost to train and deploy a model as you would other built-in Amazon SageMaker algorithms. Using the current version of XGBoost as a built-in algorithm will be familiar to users who have used the original XGBoost Release 0.72 version with the Amazon SageMaker Python SDK and want to continue using the same procedures.

```python
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
# get the URI for new container
container = get_image_uri(boto3.Session().region_name,
                          'xgboost',
                          '0.90-1');
estimator = sagemaker.estimator.Estimator(container, role, instance_count, instance_type,
                                           train_volume_size, output_path, sagemaker.Session());
```
Use Built-in Algorithms

```python
estimator.fit({'train':'s3://my-bucket/training', 'validation':'s3://my-bucket/validation})
```

If customers do not specify version in the `get_image_uri` function, they get the XGBoost Release 0.72 (p. 434) version by default. If you want to migrate to the current version, you have to specify `repo_version='0.90-1'` in the `get_image_uri` function. If you use the current version, you must update your code to use the new hyperparameters that are required by the 0.90 version of upstream algorithm. The AWS SDK for Python (Boto 3) and the CLI usage is similar. You also have to choose the version you want to run when using the Console to select the XGBoost algorithm.

**Input/Output Interface for the XGBoost Algorithm**

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training `ContentType`, valid inputs are `text/libsvm` (default) or `text/csv`.
- For Inference `ContentType`, valid inputs are `text/libsvm` or (the default) `text/csv`.

**Note**

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: `<label> <index0>:<value0> <index1>:<value1> ...`. Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

**To use a model trained with SageMaker XGBoost in open source XGBoost**

- Use the following Python code:

```python
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

**To differentiate the importance of labelled data points use Instance Weight Supports**

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For `text/libsvm` input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1 ...`. For `text/csv` input, customers need to turn on the `csv_weights`
flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

**EC2 Instance Recommendation for the XGBoost Algorithm**

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

**XGBoost Sample Notebooks**

For a sample notebook that shows how to use Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see Regression with the Amazon SageMaker XGBoost algorithm. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its **Use** tab and select **Create copy**.

**How XGBoost Works**

**XGBoost** is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using **gradient boosting** for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

**For more detail on XGBoost, see:**

- XGBoost: A Scalable Tree Boosting System
- Introduction to Boosted Trees

**XGBoost Hyperparameters**

The following table contains the subset of hyperparameters that are required or most commonly used for the Amazon SageMaker XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source DLMC XGBoost package. Currently Amazon SageMaker supports version 0.90. For details about full set of hyperparameter that can be configured for this version of XGBoost, see XGBoost Parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_round</td>
<td>The number of rounds to run the training.</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| **num_class**     | The number of classes.  
**Required** if objective is set to `multi:softmax` or `multi:softprob`.  
Valid values: integer  
Default value: |
| **alpha**         | L1 regularization term on weights. Increasing this value makes models more conservative.  
**Optional**  
Valid values: float  
Default value: 0 |
| **base_score**    | The initial prediction score of all instances, global bias.  
**Optional**  
Valid values: float  
Default value: 0.5 |
| **booster**       | Which booster to use. The `gbtree` and `dart` values use a tree-based model, while `gblinear` uses a linear function.  
**Optional**  
Valid values: String. One of `gbtree`, `gblinear`, or `dart`.  
Default value: `gbtree` |
| **colsample_bylevel** | Subsample ratio of columns for each split, in each level.  
**Optional**  
Valid values: Float. Range: [0,1].  
Default value: 1 |
| **colsample_bynode** | Subsample ratio of columns from each node.  
**Optional**  
Valid values: Float. Range: (0,1].  
Default value: 1 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colsample_bytree</td>
<td>Subsample ratio of columns when constructing each tree.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>csv_weights</td>
<td>When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>early_stopping_rounds</td>
<td>The model trains until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training. Amazon SageMaker hosting uses the best model for inference.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>eta</td>
<td>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.3</td>
</tr>
<tr>
<td>eval_metric</td>
<td>Evaluation metrics for validation data. A default metric is assigned according to the objective:</td>
</tr>
<tr>
<td></td>
<td>• rmse: for regression</td>
</tr>
<tr>
<td></td>
<td>• error: for classification</td>
</tr>
<tr>
<td></td>
<td>• map: for ranking</td>
</tr>
<tr>
<td></td>
<td>For a list of valid inputs, see XGBoost Parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: Default according to objective.</td>
</tr>
</tbody>
</table>

427
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,∞).</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>grow_policy</td>
<td>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>depthwise</code></td>
</tr>
<tr>
<td>lambda</td>
<td>L2 regularization term on weights. Increasing this value makes models more conservative.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>lambda_bias</td>
<td>L2 regularization term on bias.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>max_bin</td>
<td>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 256</td>
</tr>
<tr>
<td>max_delta_step</td>
<td>Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer. Range: [0,∞).</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_depth</td>
<td>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer. Range: [0,∞)</td>
</tr>
<tr>
<td></td>
<td>Default value: 6</td>
</tr>
<tr>
<td>max_leaves</td>
<td>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>min_child_weight</td>
<td>Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than <code>min_child_weight</code>, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,∞).</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>normalize_type</td>
<td>Type of normalization algorithm.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either <code>tree</code> or <code>forest</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>tree</code></td>
</tr>
<tr>
<td>nthread</td>
<td>Number of parallel threads used to run <code>xgboost</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: Maximum number of threads.</td>
</tr>
<tr>
<td>objective</td>
<td>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:logistic</code>, <code>multi:softmax</code>, <code>reg:squarederror</code>. For a full list of valid inputs, refer to XGBoost Parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>reg:squarederror</code></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| one_drop       | When this flag is enabled, at least one tree is always dropped during the dropout.  
  
  **Optional**  
  
  Valid values: 0 or 1  
  
  Default value: 0 |
| process_type   | The type of boosting process to run.  
  
  **Optional**  
  
  Valid values: String. Either default or update.  
  
  Default value: default |
| rate_drop      | The dropout rate that specifies the fraction of previous trees to drop during the dropout.  
  
  **Optional**  
  
  Valid values: Float. Range: [0.0, 1.0].  
  
  Default value: 0.0 |
| refresh_leaf   | This is a parameter of the 'refresh' updater plug-in. When set to true (1), tree leaves and tree node stats are updated. When set to false(0), only tree node stats are updated.  
  
  **Optional**  
  
  Valid values: 0/1  
  
  Default value: 1 |
| sample_type    | Type of sampling algorithm.  
  
  **Optional**  
  
  Valid values: Either uniform or weighted.  
  
  Default value: uniform |
| scale_pos_weight| Controls the balance of positive and negative weights. It’s useful for unbalanced classes. A typical value to consider: \( \frac{\text{sum(negative cases)}}{\text{sum(positive cases)}} \).  
  
  **Optional**  
  
  Valid values: float  
  
  Default value: 1 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>Random number seed.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
<tr>
<td>silent</td>
<td>0 means print running messages, 1 means silent mode.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
<tr>
<td>sketch_eps</td>
<td>Used only for approximate greedy algorithm. This translates into $O(1 / \text{sketch_eps})$ number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float, Range: [0, 1].</td>
</tr>
<tr>
<td>Default value: 0.03</td>
<td></td>
</tr>
<tr>
<td>skip_drop</td>
<td>Probability of skipping the dropout procedure during a boosting iteration.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td>Default value: 0.0</td>
<td></td>
</tr>
<tr>
<td>subsample</td>
<td>Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float. Range: [0, 1].</td>
</tr>
<tr>
<td>Default value: 1</td>
<td></td>
</tr>
<tr>
<td>tree_method</td>
<td>The tree construction algorithm used in XGBoost.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: One of auto, exact, approx, or hist.</td>
</tr>
<tr>
<td>Default value: auto</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tweedie_variance_power</td>
<td>Parameter that controls the variance of the Tweedie distribution.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: (1, 2).</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.5</td>
</tr>
<tr>
<td>updater</td>
<td>A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees.</td>
</tr>
<tr>
<td></td>
<td>For a full list of valid inputs, please refer to XGBoost Parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: comma-separated string.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>grow_colmaker, prune</code></td>
</tr>
</tbody>
</table>

**Tune an XGBoost Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

**Note**

Automatic model tuning for XGBoost 0.90 is only available from the SDKs, not from the Amazon SageMaker console.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

**Metrics Computed by the XGBoost Algorithm**

The XGBoost algorithm computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:accuracy</td>
<td>Classification rate, calculated as #(right)/#(all cases).</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:auc</td>
<td>Area under the curve.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:error</td>
<td>Binary classification error rate, calculated as #(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:f1</td>
<td>Indicator of classification accuracy, calculated as the harmonic mean of precision and recall.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:logloss</td>
<td>Negative log-likelihood.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mae</td>
<td>Mean absolute error.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:map</td>
<td>Mean average precision.</td>
<td>Maximize</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Optimization Direction</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>validation:merror</td>
<td>Multiclass classification error rate, calculated as #/(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mlogloss</td>
<td>Negative log-likelihood for multiclass classification.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mse</td>
<td>Mean squared error.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:ndcg</td>
<td>Normalized Discounted Cumulative Gain.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:rmse</td>
<td>Root mean square error.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

**Tunable XGBoost Hyperparameters**

Tune the open-source XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on XGBoost objective metrics are: alpha, min_child_weight, subsample, eta, and num_round.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 1000</td>
</tr>
<tr>
<td>colsample_bylevel</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 1</td>
</tr>
<tr>
<td>colsample_bynode</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 1</td>
</tr>
<tr>
<td>colsample_bytree</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.5, MaxValue: 1</td>
</tr>
<tr>
<td>eta</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 0.5</td>
</tr>
<tr>
<td>gamma</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 5</td>
</tr>
<tr>
<td>lambda</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 1000</td>
</tr>
<tr>
<td>max_delta_step</td>
<td>IntegerParameterRanges</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>max_depth</td>
<td>IntegerParameterRanges</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>min_child_weight</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 120</td>
</tr>
<tr>
<td>num_round</td>
<td>IntegerParameterRanges</td>
<td>[1, 4000]</td>
</tr>
<tr>
<td>subsample</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.5, MaxValue: 1</td>
</tr>
</tbody>
</table>

**XGBoost Previous Versions**

This page contains links to the documentation for previous versions of Amazon SageMaker XGBoost.
XGBoost Release 0.72

This previous release of the Amazon SageMaker XGBoost algorithm is based on the 0.72 release. XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and because of the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

Customers should consider using the new release of XGBoost Algorithm (p. 422). They can use it as an Amazon SageMaker built-in algorithm or as a framework to run scripts in their local environments as they would typically, for example, do with a Tensorflow deep learning framework. The new implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics. The earlier implementation of XGBoost remains available to customers if they need to postpone migrating to the new version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

Input/Output Interface for the XGBoost Release 0.72

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are text/libsvm (default) or text/csv.
- For Inference ContentType, valid inputs are text/libsvm or (the default) text/csv.

Note

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the InstanceType) must be able to hold the training dataset. For libsvm training input mode, it’s not required, but we recommend it.
SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

**To use a model trained with SageMaker XGBoost in open source XGBoost**

- Use the following Python code:

  ```python
  import pickle as pkl
  model = pkl.load(open(model_file_path, 'rb'))
  # prediction with test data
  pred = model.predict(dtest)
  ```

**To differentiate the importance of labelled data points use Instance Weight Supports**

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For text/libsvm input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...` For text/csv input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

**EC2 Instance Recommendation for the XGBoost Release 0.72**

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

**XGBoost Release 0.72 Sample Notebooks**

For a sample notebook that shows how to use the latest version of Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see [Regression with Amazon SageMaker XGBoost algorithm](#). To use the 0.72 version of XGBoost, you need to change the version in the sample code to 0.72. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances (p. 201)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its Use tab and select Create copy.

**XGBoost Release 0.72 Hyperparameters**

The following table contains the hyperparameters for the XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source XGBoost package. Currently Amazon SageMaker supports version 0.72. For more detail about hyperparameter configuration for this version of XGBoost, see [XGBoost Parameters](#).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_class</td>
<td>The number of classes.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong> if objective is set to <code>multi:softmax</code> or <code>multi:softprob</code>.</td>
</tr>
</tbody>
</table>

435
### Parameter Name | Description
--- | ---
| **num_round** | The number of rounds to run the training.  
**Required**  
Valid values: integer |
| **alpha** | L1 regularization term on weights. Increasing this value makes models more conservative.  
**Optional**  
Valid values: float  
Default value: 0 |
| **base_score** | The initial prediction score of all instances, global bias.  
**Optional**  
Valid values: float  
Default value: 0.5 |
| **booster** | Which booster to use. The gbtree and dart values use a tree-based model, while gblinear uses a linear function.  
**Optional**  
Valid values: String. One of gbtree, gblinear, or dart.  
Default value: gbtree |
| **colsample_bylevel** | Subsample ratio of columns for each split, in each level.  
**Optional**  
Valid values: Float. Range: [0,1].  
Default value: 1 |
| **colsample_bytree** | Subsample ratio of columns when constructing each tree.  
**Optional**  
Valid values: Float. Range: [0,1].  
Default value: 1 |
| **csv_weights** | When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.  
**Optional**  
Valid values: 0 or 1  
Default value: 0 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Optional</th>
<th>Valid values:</th>
<th>Default value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>early_stopping_rounds</td>
<td>The model trains until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training. Amazon SageMaker hosting uses the best model for inference.</td>
<td></td>
<td>integer</td>
<td>-</td>
</tr>
<tr>
<td>eta</td>
<td>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.</td>
<td></td>
<td>Float. Range: [0,1].</td>
<td>0.3</td>
</tr>
<tr>
<td>eval_metric</td>
<td>Evaluation metrics for validation data. A default metric is assigned according to the objective:</td>
<td></td>
<td>string</td>
<td>Default according to objective.</td>
</tr>
</tbody>
</table>
|                          | • rmse: for regression  
|                          | • error: for classification  
|                          | • map: for ranking  

For a list of valid inputs, see XGBoost Parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |          |              |               |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **lambda**     | L2 regularization term on weights. Increasing this value makes models more conservative.  
Optional  
Valid values: float  
Default value: 1 |
| **lambda_bias**| L2 regularization term on bias.  
Optional  
Valid values: Float. Range: [0.0, 1.0].  
Default value: 0 |
| **max_bin**    | Maximum number of discrete bins to bucket continuous features. Used only if `tree_method` is set to `hist`.  
Optional  
Valid values: integer  
Default value: 256 |
| **max_delta_step** | Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.  
Optional  
Valid values: Integer. Range: [0,∞).  
Default value: 0 |
| **max_depth**  | Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when `grow_policy`=depth-wise.  
Optional  
Valid values: Integer. Range: [0,∞)  
Default value: 6 |
| **max_leaves** | Maximum number of nodes to be added. Relevant only if `grow_policy` is set to lossguide.  
Optional  
Valid values: integer  
Default value: 0 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>min_child_weight</strong></td>
<td>Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than <strong>min_child_weight</strong>, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is. <strong>Optional</strong> Valid values: Float. Range: [0,∞). Default value: 1</td>
</tr>
<tr>
<td><strong>normalize_type</strong></td>
<td>Type of normalization algorithm. <strong>Optional</strong> Valid values: Either <code>tree</code> or <code>forest</code>. Default value: <code>tree</code></td>
</tr>
<tr>
<td><strong>nthread</strong></td>
<td>Number of parallel threads used to run <code>xgboost</code>. <strong>Optional</strong> Valid values: integer  Default value: Maximum number of threads.</td>
</tr>
<tr>
<td><strong>objective</strong></td>
<td>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:logistic</code>, <code>reg:softmax</code>, <code>multi:squarederror</code>. For a full list of valid inputs, refer to <code>XGBoost Parameters</code>. <strong>Optional</strong> Valid values: string  Default value: <code>reg:squarederror</code></td>
</tr>
<tr>
<td><strong>one_drop</strong></td>
<td>When this flag is enabled, at least one tree is always dropped during the dropout. <strong>Optional</strong> Valid values: 0 or 1  Default value: 0</td>
</tr>
<tr>
<td><strong>process_type</strong></td>
<td>The type of boosting process to run. <strong>Optional</strong> Valid values: String. Either <code>default</code> or <code>update</code>.  Default value: <code>default</code></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rate_drop</td>
<td>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>refresh_leaf</td>
<td>This is a parameter of the 'refresh' updater plug-in. When set to true (1), tree leaves and tree node stats are updated. When set to false (0), only tree node stats are updated.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0/1</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>sample_type</td>
<td>Type of sampling algorithm.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either uniform or weighted.</td>
</tr>
<tr>
<td></td>
<td>Default value: uniform</td>
</tr>
<tr>
<td>scale_pos_weight</td>
<td>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: ( \frac{\text{sum(negative cases)}}{\text{sum(positive cases)}} ).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>seed</td>
<td>Random number seed.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>silent</td>
<td>0 means print running messages, 1 means silent mode.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `sketch_eps`   | Used only for approximate greedy algorithm. This translates into $O(1 / \text{sketch}\_\text{eps})$ number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.  
Optional  
Valid values: Float, Range: [0, 1].  
Default value: 0.03 |
| `skip_drop`    | Probability of skipping the dropout procedure during a boosting iteration.  
Optional  
Valid values: Float. Range: [0.0, 1.0].  
Default value: 0.0 |
| `subsample`    | Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.  
Optional  
Valid values: Float. Range: [0,1].  
Default value: 1 |
| `tree_method`  | The tree construction algorithm used in XGBoost.  
Optional  
Valid values: One of auto, exact, approx, or hist.  
Default value: auto |
| `tweedie_variance_power` | Parameter that controls the variance of the Tweedie distribution.  
Optional  
Valid values: Float. Range: (1, 2).  
Default value: 1.5 |
| `updater`      | A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees.  
For a full list of valid inputs, please refer to XGBoost Parameters.  
Optional  
Valid values: comma-separated string.  
Default value: `grow_colmaker, prune` |
Tune an XGBoost Release 0.72 Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Perform Automatic Model Tuning (p. 555).

Metrics Computed by the XGBoost Release 0.72 Algorithm

The XGBoost algorithm based on version 0.72 computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:auc</td>
<td>Area under the curve.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:error</td>
<td>Binary classification error rate, calculated as #(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:logloss</td>
<td>Negative log-likelihood.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mae</td>
<td>Mean absolute error.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:map</td>
<td>Mean average precision.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:merror</td>
<td>Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mlogloss</td>
<td>Negative log-likelihood for multiclass classification.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:ndcg</td>
<td>Normalized Discounted Cumulative Gain.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:rmse</td>
<td>Root mean square error.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Tunable XGBoost Release 0.72 Hyperparameters

Tune the XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on XGBoost objective metrics are: alpha, min_child_weight, subsample, eta, and num_round.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 1000</td>
</tr>
<tr>
<td>colsample_bylevel</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 1</td>
</tr>
<tr>
<td>colsample_bytree</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.5, MaxValue: 1</td>
</tr>
<tr>
<td>eta</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 0.5</td>
</tr>
</tbody>
</table>
Use Machine Learning Frameworks with Amazon SageMaker

The Amazon SageMaker Python SDK provides open source APIs and containers that make it easy to train and deploy models in Amazon SageMaker with several different machine learning and deep learning frameworks. For general information about the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk. For information about using specific frameworks in Amazon SageMaker, see the following topics:

Topics
- Use Apache Spark with Amazon SageMaker (p. 443)
- Use TensorFlow with Amazon SageMaker (p. 452)
- Use Apache MXNet with Amazon SageMaker (p. 453)
- Use Scikit-learn with Amazon SageMaker (p. 453)
- Use PyTorch with Amazon SageMaker (p. 454)
- Use Chainer with Amazon SageMaker (p. 455)
- Use SparkML Serving with Amazon SageMaker (p. 455)

Use Apache Spark with Amazon SageMaker

This section provides information for developers who want to use Apache Spark for preprocessing data and Amazon SageMaker for model training and hosting. For information about supported versions of Apache Spark, see https://github.com/aws/sagemaker-spark#getting-sagemaker-spark.

Amazon SageMaker provides an Apache Spark library, in both Python and Scala, that you can use to easily train models in Amazon SageMaker using org.apache.spark.sql.DataFrame data frames in your Spark clusters. After model training, you can also host the model using Amazon SageMaker hosting services.

The Amazon SageMaker Spark library, com.amazonaws.services.sagemaker.sparksdk, provides the following classes, among others:

- SageMakerEstimator—Extends the org.apache.spark.ml.Estimator interface. You can use this estimator for model training in Amazon SageMaker.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 5</td>
</tr>
<tr>
<td>lambda</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 1000</td>
</tr>
<tr>
<td>max_delta_step</td>
<td>IntegerParameterRanges</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>max_depth</td>
<td>IntegerParameterRanges</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>min_child_weight</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0, MaxValue: 120</td>
</tr>
<tr>
<td>num_round</td>
<td>IntegerParameterRanges</td>
<td>[1, 4000]</td>
</tr>
<tr>
<td>subsample</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.5, MaxValue: 1</td>
</tr>
</tbody>
</table>
• KMeansSageMakerEstimator, PCA SageMakerEstimator, and XGBoostSageMakerEstimator—Extend the SageMakerEstimator class.

• SageMakerModel—Extends the org.apache.spark.ml.Model class. You can use this SageMakerModel for model hosting and obtaining inferences in Amazon SageMaker.

Download the Amazon SageMaker Spark Library

You have the following options for downloading the Spark library provided by Amazon SageMaker:

• You can download the source code for both PySpark and Scala libraries from GitHub at https://github.com/aws/sagemaker-spark.

• For the Python Spark library, you have the following additional options:
  • Use pip install:
    ```
    # pip install sagemaker_pyspark
    ```
  • In a notebook instance, create a new notebook that uses either the Sparkmagic (PySpark) or the Sparkmagic (PySpark3) kernel and connect to a remote Amazon EMR cluster. For more information, see Build Amazon SageMaker Notebooks Backed by Spark in Amazon EMR.

  **Note**
  The EMR cluster must be configured with an IAM role that has the AmazonSageMakerFullAccess policy attached. For information about configuring roles for an EMR cluster, see Configure IAM Roles for Amazon EMR Permissions to AWS Services in the Amazon EMR Management Guide.

• You can get the Scala library from Maven. Add the Spark library to your project by adding the following dependency to your pom.xml file:

```xml
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>sagemaker-spark_2.11</artifactId>
  <version>spark_2.2.0-1.0</version>
</dependency>
```

Integrate Your Apache Spark Application with Amazon SageMaker

The following is high-level summary of the steps for integrating your Apache Spark application with Amazon SageMaker.

1. Continue data preprocessing using the Apache Spark library that you are familiar with. Your dataset remains a DataFrame in your Spark cluster.

  **Note**
  Load your data into a DataFrame and preprocess it so that you have a features column with org.apache.spark.ml.linalg.Vector of Doubles, and an optional label column with values of Double type.

2. Use the estimator in the Amazon SageMaker Spark library to train your model. For example, if you choose the k-means algorithm provided by Amazon SageMaker for model training, you call the KMeansSageMakerEstimator.fit method.

  Provide your DataFrame as input. The estimator returns a SageMakerModel object.
Note
SageMakerModel extends the org.apache.spark.ml.Model.

The fit method does the following:

a. Converts the input DataFrame to the protobuf format by selecting the features and label columns from the input DataFrame and uploading the protobuf data to an Amazon S3 bucket. The protobuf format is efficient for model training in Amazon SageMaker.

b. Starts model training in Amazon SageMaker by sending an Amazon SageMaker CreateTrainingJob (p. 931) request. After model training has completed, Amazon SageMaker saves the model artifacts to an S3 bucket.

Amazon SageMaker assumes the IAM role that you specified for model training to perform tasks on your behalf. For example, it uses the role to read training data from an S3 bucket and to write model artifacts to a bucket.

c. Creates and returns a SageMakerModel object. The constructor does the following tasks, which are related to deploying your model to Amazon SageMaker.

i. Sends a CreateModel (p. 902) request to Amazon SageMaker.

ii. Sends a CreateEndpointConfig (p. 878) request to Amazon SageMaker.

iii. Sends a CreateEndpoint (p. 875) request to Amazon SageMaker, which then launches the specified resources, and hosts the model on them.

3. You can get inferences from your model hosted in Amazon SageMaker with the SageMakerModel.transform.

Provide an input DataFrame with features as input. The transform method transforms it to a DataFrame containing inferences. Internally, the transform method sends a request to the InvokeEndpoint (p. 1260) Amazon SageMaker API to get inferences. The transform method appends the inferences to the input DataFrame.

Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark

Topics

- Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark (p. 449)
- Use the SageMakerEstimator in a Spark Pipeline (p. 450)

Amazon SageMaker provides an Apache Spark library (in both Python and Scala) that you can use to integrate your Apache Spark applications with Amazon SageMaker. For example, you might use Apache Spark for data preprocessing and Amazon SageMaker for model training and hosting. For more information, see Use Apache Spark with Amazon SageMaker (p. 443). This section provides example code that uses the Apache Spark Scala library provided by Amazon SageMaker to train a model in Amazon SageMaker using DataFrames in your Spark cluster. The example also hosts the resulting model artifacts using Amazon SageMaker hosting services. Specifically, this example does the following:

- Uses the KMeansSageMakerEstimator to fit (or train) a model on data

Because the example uses the k-means algorithm provided by Amazon SageMaker to train a model, you use the KMeansSageMakerEstimator. You train the model using images of handwritten single-digit numbers (from the MNIST dataset). You provide the images as an input DataFrame. For your convenience, Amazon SageMaker provides this dataset in an S3 bucket.
In response, the estimator returns a `SageMakerModel` object.

- Obtains inferences using the trained `SageMakerModel`

To get inferences from a model hosted in Amazon SageMaker, you call the `SageMakerModel.transform` method. You pass a `DataFrame` as input. The method transforms the input `DataFrame` to another `DataFrame` containing inferences obtained from the model.

For a given input image of a handwritten single-digit number, the inference identifies a cluster that the image belongs to. For more information, see K-Means Algorithm (p. 307).

This is the example code:

```scala
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator
val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")
val roleArn = "arn:aws:iam::account-id:role/rolename"
val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1
).setK(10).setFeatureDim(784)

// train
val model = estimator.fit(trainingData)
val transformedData = model.transform(testData)
transformedData.show
```

The code does the following:

- Loads the MNIST dataset from an S3 bucket provided by Amazon SageMaker (awsai-sparksdk-dataset) into a Spark DataFrame (`mnistTrainingDataFrame`):
// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

val roleArn = "arn:aws:iam::account-id:role/rolename"
trainingData.show()

The `show` method displays the first 20 rows in the data frame:

+-----+--------------------+
| label|            features |
+-----+--------------------+
|  5.0 | (784,[152,153,154...|
|  0.0 | (784,[127,128,129...|
|  4.0 | (784,[160,161,162...|
|  1.0 | (784,[158,159,160...|
|  9.0 | (784,[208,209,210...|
|  2.0 | (784,[155,156,157...|
|  1.0 | (784,[124,125,126...|
|  3.0 | (784,[151,152,153...|
|  1.0 | (784,[152,153,154...|
|  4.0 | (784,[134,135,161...|
|  3.0 | (784,[123,124,125...|
|  5.0 | (784,[216,217,218...|
|  3.0 | (784,[143,144,145...|
|  6.0 | (784,[72,73,74,99...|
|  1.0 | (784,[151,152,153...|
|  7.0 | (784,[211,212,213...|
|  2.0 | (784,[151,152,153...|
|  8.0 | (784,[159,160,161...|
|  6.0 | (784,[100,101,102...|
|  9.0 | (784,[209,210,211...|
+-----+--------------------+

only showing top 20 rows

In each row:

- The `label` column identifies the image's label. For example, if the image of the handwritten number is the digit 5, the label value is 5.
- The `features` column stores a vector (`org.apache.spark.ml.linalg.Vector`) of Double values. These are the 784 features of the handwritten number. (Each handwritten number is a 28 x 28-pixel image, making 784 features.)

- Creates an Amazon SageMaker estimator (`KMeansSageMakerEstimator`)

The `fit` method of this estimator uses the k-means algorithm provided by Amazon SageMaker to train models using an input `DataFrame`. In response, it returns a `SageMakerModel` object that you can use to get inferences.

**Note**

The `KMeansSageMakerEstimator` extends the Amazon SageMaker `SageMakerEstimator`, which extends the Apache Spark Estimator.

val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
trainingInstanceType = "ml.p2.xlarge",
trainingInstanceCount = 1,
endpointInstanceType = "ml.c4.xlarge",
endpointInitialInstanceCount = 1)
.setK(10).setFeatureDim(784)

The constructor parameters provide information that is used for training a model and deploying it on
Amazon SageMaker:
• trainingInstanceType and trainingInstanceCount—Identify the type and number of ML
compute instances to use for model training.

• endpointInstanceType—Identifies the ML compute instance type to use when hosting the model
in Amazon SageMaker. By default, one ML compute instance is assumed.

• endpointInitialInstanceCount—Identifies the number of ML compute instances initially
backing the endpoint hosting the model in Amazon SageMaker.

• sagemakerRole—Amazon SageMaker assumes this IAM role to perform tasks on your behalf. For
example, for model training, it reads data from S3 and writes training results (model artifacts) to S3.

Note
This example implicitly creates an Amazon SageMaker client. To create this client, you must
provide your credentials. The API uses these credentials to authenticate requests to Amazon
SageMaker. For example, it uses the credentials to authenticate requests to create a training
job and API calls for deploying the model using Amazon SageMaker hosting services.

• After the KMeansSageMakerEstimator object has been created, you set the following parameters,
are used in model training:
• The number of clusters that the k-means algorithm should create during model training. You
specify 10 clusters, one for each digit, 0 through 9.
• Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel
image, making 784 features).

• Calls the estimator fit method

// train
val model = estimator.fit(trainingData)

You pass the input DataFrame as a parameter. The model does all the work of training the model
and deploying it to Amazon SageMaker. For more information see, Integrate Your Apache Spark
Application with Amazon SageMaker (p. 444). In response, you get a SageMakerModel object, which
you can use to get inferences from your model deployed in Amazon SageMaker.

You provide only the input DataFrame. You don't need to specify the registry path to the k-means
algorithm used for model training because the KMeansSageMakerEstimator knows it.

• Calls the SageMakerModel.transform method to get inferences from the model deployed in
Amazon SageMaker.
The `transform` method takes a DataFrame as input, transforms it, and returns another DataFrame containing inferences obtained from the model.

```scala
val transformedData = model.transform(testData)
transformedData.show
```

For simplicity, we use the same DataFrame as input to the `transform` method that we used for model training in this example. The `transform` method does the following:

- Serializes the `features` column in the input DataFrame to protobuf and sends it to the Amazon SageMaker endpoint for inference.
- Deserializes the protobuf response into the two additional columns (`distance_to_cluster` and `closest_cluster`) in the transformed DataFrame.

The `show` method gets inferences to the first 20 rows in the input DataFrame:

```
<table>
<thead>
<tr>
<th>label</th>
<th>features</th>
<th>distance_to_cluster</th>
<th>closest_cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>(784,[152,153,154,...</td>
<td>1767.897705078125</td>
<td>4.0</td>
</tr>
<tr>
<td>0.0</td>
<td>(784,[127,128,129,...</td>
<td>1392.157470703125</td>
<td>5.0</td>
</tr>
<tr>
<td>4.0</td>
<td>(784,[160,161,162,...</td>
<td>1671.5711669921875</td>
<td>9.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[158,159,160,...</td>
<td>1182.6082763671875</td>
<td>6.0</td>
</tr>
<tr>
<td>9.0</td>
<td>(784,[208,209,210,...</td>
<td>1390.4002685546875</td>
<td>0.0</td>
</tr>
<tr>
<td>2.0</td>
<td>(784,[155,156,157,...</td>
<td>1713.988037109375</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[124,125,126,...</td>
<td>1246.3016357421875</td>
<td>2.0</td>
</tr>
<tr>
<td>3.0</td>
<td>(784,[151,152,153,...</td>
<td>1753.229248046875</td>
<td>4.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[152,153,154,...</td>
<td>978.8394165039062</td>
<td>2.0</td>
</tr>
<tr>
<td>4.0</td>
<td>(784,[134,135,161,...</td>
<td>1623.176513671875</td>
<td>3.0</td>
</tr>
<tr>
<td>3.0</td>
<td>(784,[123,124,125,...</td>
<td>1533.863525390625</td>
<td>4.0</td>
</tr>
<tr>
<td>5.0</td>
<td>(784,[216,217,218,...</td>
<td>1469.357177734375</td>
<td>6.0</td>
</tr>
<tr>
<td>3.0</td>
<td>(784,[143,144,145,...</td>
<td>1736.765689140625</td>
<td>4.0</td>
</tr>
<tr>
<td>6.0</td>
<td>(784,[72,73,74,99,...</td>
<td>1473.69384765625</td>
<td>8.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[151,152,153,...</td>
<td>944.88720703125</td>
<td>2.0</td>
</tr>
<tr>
<td>7.0</td>
<td>(784,[211,212,213,...</td>
<td>1285.9071044921875</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0</td>
<td>(784,[151,152,153,...</td>
<td>1635.0125732421875</td>
<td>1.0</td>
</tr>
<tr>
<td>8.0</td>
<td>(784,[159,160,161,...</td>
<td>1436.3162841796875</td>
<td>6.0</td>
</tr>
<tr>
<td>6.0</td>
<td>(784,[100,101,102,...</td>
<td>1499.7366943359375</td>
<td>7.0</td>
</tr>
<tr>
<td>9.0</td>
<td>(784,[209,210,211,...</td>
<td>1564.6319580078125</td>
<td>6.0</td>
</tr>
</tbody>
</table>
```

You can interpret the data, as follows:

- A handwritten number with the label 5 belongs to cluster 5 (`closest_cluster`).
- A handwritten number with the label 0 belongs to cluster 2.
- A handwritten number with the label 4 belongs to cluster 4.
- A handwritten number with the label 1 belongs to cluster 1.

For more information on how to run these examples, see https://github.com/aws/sagemaker-spark/blob/master/README.md on GitHub.

**Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark**

In Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark (p. 445), you use the `kMeansSageMakerEstimator` because the example uses the k-means algorithm provided by Amazon SageMaker for model training. You might choose to use your own custom algorithm for model...
training instead. Assuming that you have already created a Docker image, you can create your own SageMakerEstimator and specify the Amazon Elastic Container Registry path for your custom image.

The following example shows how to create a KMeansSageMakerEstimator from the SageMakerEstimator. In the new estimator, you explicitly specify the Docker registry path to your training and inference code images.

```scala
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.SageMakerEstimator
import com.amazonaws.services.sagemaker.sparksdk.transformation.serializers.ProtobufRequestRowSerializer
import com.amazonaws.services.sagemaker.sparksdk.transformation.deserializers.KMeansProtobufResponseRowDeserializer

val estimator = new SageMakerEstimator(
  trainingImage = "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  modelImage = "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  requestRowSerializer = new ProtobufRequestRowSerializer(),
  responseRowDeserializer = new KMeansProtobufResponseRowDeserializer(),
  hyperParameters = Map("k" -> "10", "feature_dim" -> "784"),
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1,
  trainingSparkDataFormat = "sagemaker")
```

In the code, the parameters in the SageMakerEstimator constructor include:

- **trainingImage** —Identifies the Docker registry path to the training image containing your custom code.
- **modelImage** —Identifies the Docker registry path to the image containing inference code.
- **requestRowSerializer** —Implements
  ```scala
  com.amazonaws.services.sagemaker.sparksdk.transformation.RequestRowSerializer.
  ```
  This parameter serializes rows in the input DataFrame to send them to the model hosted in Amazon SageMaker for inference.
- **responseRowDeserializer** —Implements
  ```scala
  com.amazonaws.services.sagemaker.sparksdk.transformation.ResponseRowDeserializer.
  ```
  This parameter deserializes responses from the model, hosted in Amazon SageMaker, back into a DataFrame.
- **trainingSparkDataFormat** —Specifies the data format that Spark uses when uploading training data from a DataFrame to S3. For example, "sagemaker" for protobuf format, "csv" for comma-separated values, and "libsvm" for LibSVM format.

You can implement your own RequestRowSerializer and ResponseRowDeserializer to serialize and deserialize rows from a data format that your inference code supports, such as .libsvm or .csv.

**Use the SageMakerEstimator in a Spark Pipeline**

You can use org.apache.spark.ml.Estimators and org.apache.spark.ml.models, and SageMakerEstimator estimators and SageMakerModel models in org.apache.spark.ml.Pipeline pipelines, as shown in the following example:

```scala
import org.apache.spark.ml.Pipeline
```
import org.apache.spark.ml.feature.PCA
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

// substitute your SageMaker IAM role here
val roleArn = "arn:aws:iam::<account-id>:role/rolename"

val pcaEstimator = new PCA()
  .setInputCol("features")
  .setOutputCol("projectedFeatures")
  .setK(50)

val kMeansSageMakerEstimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(integTestingRole),
  requestRowSerializer = new ProtobufRequestRowSerializer(featuresColumnName = "projectedFeatures"),
  trainingSparkDataFormatOptions = Map("featuresColumnName" -> "projectedFeatures"),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(50)

val pipeline = new Pipeline().setStages(Array(pcaEstimator, kMeansSageMakerEstimator))

// train
val pipelineModel = pipeline.fit(trainingData)
val transformedData = pipelineModel.transform(testData)
transformedData.show()

The parameter trainingSparkDataFormatOptions configures Spark to serialize to protobuf the "projectedFeatures" column for model training. Additionally, Spark serializes to protobuf the "label" column by default.

Because we want to make inferences using the "projectedFeatures" column, we pass the column name into the ProtobufRequestRowSerializer.

The following example shows a transformed DataFrame:

```
+-----+--------------------+--------------------+-------------------+---------------+
|label|            features|   projectedFeatures|distance_to_cluster|closest_cluster|
+-----+--------------------+--------------------+-------------------+---------------+
|  5.0| [784, [152, 153, 154... | [880.731433034386386| 1500.470703125| 0.0|
|  0.0| [784, [127, 128, 129... | [1142.18359375| 1142.18359375| 4.0|
|  4.0| [784, [160, 161, 162... | [704.9492362931406| 1386.246826171875| 9.0|
|  1.0| [784, [158, 159, 160... | [-42.3281921937712| 1277.0736083984375| 5.0|
|  9.0| [784, [208, 209, 210... | [374.0439020283334| 1211.0092734375| 3.0|
|  2.0| [784, [155, 156, 157... | [941.2677145288505| 1496.15795884375| 8.0|
|  1.0| [784, [124, 125, 126... | [30.28485964105949| 1327.6766357421875| 5.0|
|  3.0| [784, [151, 152, 153... | [1270.14374062052| 1570.7674560546875| 0.0|
```
### Additional Examples: Use Amazon SageMaker with Apache Spark

Additional examples of using Amazon SageMaker with Apache Spark are available at [https://github.com/aws/sagemaker-spark/tree/master/examples](https://github.com/aws/sagemaker-spark/tree/master/examples).

### Use TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom TensorFlow code. The Amazon SageMaker Python SDK TensorFlow estimators and models and the Amazon SageMaker open-source TensorFlow containers make writing a TensorFlow script and running it in Amazon SageMaker easier.

### Use TensorFlow Version 1.11 and Later

For TensorFlow versions 1.11 and later, the Amazon SageMaker Python SDK supports script mode training scripts.

### What do you want to do?

I want to train a custom TensorFlow model in Amazon SageMaker.


For documentation, see [Train a Model with TensorFlow](https://docs.aws.amazon.com/sagemaker/latest/dg/tensorflow-training.html).

I have a TensorFlow model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

**Deploy TensorFlow Serving models.**

I have a TensorFlow model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint.

**Deploying directly from model artifacts.**

I want to see the API documentation for Amazon SageMaker Python SDK TensorFlow classes.

**TensorFlow Estimator**

I want to see information about Amazon SageMaker TensorFlow containers.


For general information about writing TensorFlow script mode training scripts and using TensorFlow script mode estimators and models with Amazon SageMaker, see [Using TensorFlow with the SageMaker Python SDK](https://docs.aws.amazon.com/sagemaker/latest/dg/tensorflow-training.html).
For information about TensorFlow versions supported by the Amazon SageMaker TensorFlow container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

Use TensorFlow Legacy Mode for Versions 1.11 and Earlier

The Amazon SageMaker Python SDK provides a legacy mode that supports TensorFlow versions 1.11 and earlier. Use legacy mode TensorFlow training scripts to run TensorFlow jobs in Amazon SageMaker if:

- You have existing legacy mode scripts that you do not want to convert to script mode.
- You want to use a TensorFlow version earlier than 1.11.

For information about writing legacy mode TensorFlow scripts to use with the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk/tree/v1.12.0/src/sagemaker/tensorflow#tensorflow-sagemaker-estimators-and-models.

Use Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom MXNet code. The Amazon SageMaker Python SDK MXNet estimators and models and the Amazon SageMaker open-source MXNet container make writing a MXNet script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom MXNet model in Amazon SageMaker.


For documentation, see Train a Model with MXNet.

I have an MXNet model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

- Deploy MXNet models.

I have an MXNet model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

- Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK MXNet classes.

- MXNet Classes

I want to see information about Amazon SageMaker MXNet containers.


For general information about writing MXNet script mode training scripts and using MXNet script mode estimators and models with Amazon SageMaker, see Using MXNet with the SageMaker Python SDK.

For information about MXNet versions supported by the Amazon SageMaker MXNet container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

Use Scikit-learn with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Scikit-learn code. The Amazon SageMaker Python SDK Scikit-learn estimators and models and the Amazon SageMaker open-
source Scikit-learn container make writing a Scikit-learn script and running it in Amazon SageMaker easier.

**What do you want to do?**

I want to use Scikit-learn for data processing, feature engineering, or model evaluation in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_processing/scikit_learn_data_processing_and_model_evaluation.

For documentation, see Amazon SageMaker Python SDK ReadTheDocs

I want to train a custom Scikit-learn model in Amazon SageMaker.


For documentation, see Train a Model with Scikit-learn.

I have a Scikit-learn model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

Deploy Scikit-learn models.

I have a Scikit-learn model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

Deploy Endpoints from Model Data.

I want to see the API documentation for Amazon SageMaker Python SDK Scikit-learn classes.

Scikit-learn Classes

I want to see information about Amazon SageMaker Scikit-learn containers.


For general information about writing Scikit-learn training scripts and using Scikit-learn estimators and models with Amazon SageMaker, see Using Scikit-learn with the SageMaker Python SDK.

For information about Scikit-learn versions supported by the Amazon SageMaker Scikit-learn container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/sklearn/README.rst.

**Use PyTorch with Amazon SageMaker**

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. The Amazon SageMaker Python SDK PyTorch estimators and models and the Amazon SageMaker open-source PyTorch container make writing a PyTorch script and running it in Amazon SageMaker easier.

**What do you want to do?**

I want to train a custom PyTorch model in Amazon SageMaker.


For documentation, see Train a Model with PyTorch.

I have a PyTorch model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

Deploy PyTorch models.
I have a PyTorch model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

Deploy Endpoints from Model Data.
I want to see the API documentation for Amazon SageMaker Python SDK PyTorch classes.

PyTorch Classes
I want to see information about Amazon SageMaker PyTorch containers.


For general information about writing PyTorch training scripts and using PyTorch estimators and models with Amazon SageMaker, see Using PyTorch with the SageMaker Python SDK.

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst.

Use Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. The Amazon SageMaker Python SDK Chainer estimators and models and the Amazon SageMaker open-source Chainer container make writing a Chainer script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom Chainer model in Amazon SageMaker.


For documentation, see Train a Model with Chainer.

I have a Chainer model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

Deploy Chainer models.
I have a Chainer model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

Deploy Endpoints from Model Data.
I want to see the API documentation for Amazon SageMaker Python SDK Chainer classes.

Chainer Classes
I want to see information about Amazon SageMaker Chainer containers.


For general information about writing Chainer training scripts and using Chainer estimators and models with Amazon SageMaker, see Using Chainer with the SageMaker Python SDK.

For information about Chainer versions supported by the Amazon SageMaker Chainer container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/chainer/README.rst.

Use SparkML Serving with Amazon SageMaker

The Amazon SageMaker Python SDK SparkML Serving model and predictor and the Amazon SageMaker open-source SparkML Serving container support deploying Apache Spark ML pipelines serialized with MLep in Amazon SageMaker to get inferences.

**Use Your Own Algorithms or Models with Amazon SageMaker**

Amazon SageMaker makes extensive use of Docker containers for build and runtime tasks. Before using your own algorithm or model with Amazon SageMaker, you need to understand how Amazon SageMaker manages and runs them. Amazon SageMaker provides pre-built Docker images for its built-in algorithms and the supported deep learning frameworks used for training and inference. By using containers, you can train machine learning algorithms and deploy models quickly and reliably at any scale. Docker is a program that performs operating-system-level virtualization for installing, distributing, and managing software. It packages applications and their dependencies into virtual containers that provide isolation, portability, and security.

You can put scripts, algorithms, and inference code for your machine learning models into containers. The container includes the runtime, system tools, system libraries, and other code required to train your algorithms or deploy your models. This gives you the flexibility to use almost any script or algorithm code with Amazon SageMaker, regardless of runtime or implementation language. The code that runs in containers is effectively isolated from its surroundings, ensuring a consistent runtime, regardless of where the container is deployed. After packaging your training code, inference code, or both into Docker containers, you can create algorithm resources and model package resources for use in Amazon SageMaker or to publish on AWS Marketplace. With Docker, you can ship code faster, standardize application operations, seamlessly move code, and economize by improving resource utilization.

You create Docker containers from images that are saved in a repository. You build the images from scripted instructions provided in a Dockerfile. To use Docker containers in Amazon SageMaker, the scripts that you use must satisfy certain requirements. For information about the requirements, see Use Your Own Training Algorithms (p. 476) and Use Your Own Inference Code (p. 480).

**Scenarios for Running Scripts, Training Algorithms, or Deploying Models with Amazon SageMaker**

Amazon SageMaker always uses Docker containers when running scripts, training algorithms or deploying models. However, your level of engagement with containers varies depending on whether you are using a built-in algorithm provided by Amazon SageMaker or a script or model that you have developed yourself. If you're using your own code, it also depends on the language and framework or environment used to develop it, and any other of the dependencies it requires to run. In particular, it depends on whether you use the Amazon SageMaker Python SDK or AWS SDK for Python (Boto3) or some other SDK. Amazon SageMaker provides containers for its built-in algorithms and pre-built Docker images for some of the most common machine learning frameworks. You can use the containers and images as provided or extend them to cover more complicated use cases. You can also create your own container images to manage more advanced use cases not addressed by the containers provided by Amazon SageMaker.

There are four main scenarios for running scripts, algorithms, and models in the Amazon SageMaker environment. The last three describe the scenarios covered here: the ways you can use containers to bring your own script, algorithm or model.

- **Use a built-in algorithm.** Containers are used behind the scenes when you use one of the Amazon SageMaker built-in algorithms, but you do not deal with them directly. You can train and deploy these algorithms from the Amazon SageMaker console, the AWS Command Line Interface (AWS CLI), a Python notebook, or the Amazon SageMaker Python SDK. The built-in algorithms available are
itemized and described in the Use Amazon SageMaker Built-in Algorithms (p. 220) topic. For an example of how to train and deploy a built-in algorithm using Jupyter Notebook running in an Amazon SageMaker notebook instance, see the Get Started with Amazon SageMaker (p. 20) topic.

- **Use pre-built container images.** Amazon SageMaker provides pre-built containers to support deep learning frameworks such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports machine learning libraries such as scikit-learn and SparkML by providing pre-built Docker images. If you use the Amazon SageMaker Python SDK, they are deployed using their respective Amazon SageMaker SDK Estimator class. In this case, you can supply the Python code that implements your algorithm and configure the pre-built image to access your code as an entry point. For a list of deep learning frameworks currently supported by Amazon SageMaker and samples that show how to use their pre-build container images, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 470). For information on the scikit-learn and SparkML pre-built container images, see Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML (p. 473). For more information about using frameworks with the Amazon SageMaker Python SDK, see their respective topics in Use Machine Learning Frameworks with Amazon SageMaker (p. 443).

- **Extend a pre-built container image.** If you have additional functional requirements for an algorithm or model that you developed in a framework that a pre-built Amazon SageMaker Docker image doesn't support, you can modify an Amazon SageMaker image to satisfy your needs. For an example, see Extending our PyTorch containers.

- **Build your own custom container image:** If there is no pre-built Amazon SageMaker container image that you can use or modify for an advanced scenario, you can package your own script or algorithm to use with Amazon SageMaker. You can use any programming language or framework to develop your container. For an example that shows how to build your own containers to train and host an algorithm, see Bring Your Own R Algorithm.

The next topic provides a brief introduction to Docker containers. Amazon SageMaker has certain contractual requirements that a container must satisfy to be used with it. The following topic describes the Amazon SageMaker Containers library that can be used to create Amazon SageMaker-compatible containers, including a list of the environmental variables it defines and may require. Then a tutorial that shows how to get started by using Amazon SageMaker Containers to train a Python script. After the tutorial, topics:

- Describe the pre-built Docker containers provided by Amazon SageMaker for deep learning frameworks and other libraries.
- Provide examples of how to deploy containers for the various scenarios.

Subsequent sections describe in more detail the contractual requirements to use Docker with Amazon SageMaker to train your custom algorithms and to deploy your inference code to make predictions. There are two ways to make predictions when deploying a model. First, to get individual, real-time predictions, you can make inferences with a hosting service. Second, to get predictions for an entire dataset, you can use a batch transform. The final sections describe how to create algorithm and model package resources for use in your Amazon SageMaker account or to publish on AWS Marketplace.

**Topics**

- Docker Container Basics (p. 458)
- Amazon SageMaker Containers: a Library to Create Docker Containers (p. 458)
- Get Started: Use Amazon SageMaker Containers to Run a Python Script (p. 467)
- Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 470)
- Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML (p. 473)
- Example Notebooks: Use Your Own Algorithm or Model (p. 475)
- Use Your Own Training Algorithms (p. 476)
- Use Your Own Inference Code (p. 480)
Docker Container Basics

Docker containers provide isolation, portability, and security. They simplify the creation of highly distributed systems and save money by improving resource utilization. Docker relies on Linux kernel functionality to provide a lightweight virtualization to package applications into an image that is totally self-contained. Docker uses a file, called a Dockerfile, to specify how the image is assembled. When you have an image, you use Docker to build and run a container based on that image.

You can build your Docker images from scratch or base them on other Docker images that you or others have built. Images are stored in repositories that are indexed and maintained by registries. An image can be pushed into or pulled out of a repository using its registry address, which is similar to a URL. Docker Hub is a registry hosted by Docker, Inc. that provides publicly available repositories. AWS provides the Amazon Elastic Container Service (Amazon ECS), a highly scalable, fast container management service. With Amazon ECS, you can deploy any kind of code in Amazon SageMaker. You can also create a logical division of labor by creating a deployment team that handles DevOps and infrastructure, and that maintains the container, and a data science team that creates the algorithms and models that are later added to a container.

Docker builds images by reading the instructions from a Dockerfile text file that contains all of the commands, in order, that are needed to build the image. A Dockerfile adheres to a specific format and set of instructions. For more information, see Dockerfile reference. Dockerfiles used in Amazon SageMaker must also satisfy additional requirements regarding the environmental variables, directory structure, timeouts, and other common functionality. For information, see Use Your Own Training Algorithms (p. 476) and Use Your Own Inference Code (p. 480).

For general information about Docker containers managed by Amazon ECS, see Docker Basics for Amazon ECS in the Amazon Elastic Container Service Developer Guide.

For more information about writing Dockerfiles to build images, see Best practices for writing Dockerfiles.

For general information about Docker, see the following:

- Docker home page
- Docker overview
- Getting Started with Docker
- Dockerfile reference

Amazon SageMaker Containers: a Library to Create Docker Containers

Amazon SageMaker Containers is a library that implements the functionality that you need to create containers to run scripts, train algorithms, or deploy models that are compatible with Amazon SageMaker. To install this library, use a `RUN pip install sagemaker-containers` command in your Dockerfile. The library defines the locations for storing code and other resources when you install it. Your Dockerfile must also copy the code to be run into the location expected by an Amazon SageMaker-compatible container and define the entry point containing the code to run when the container is started. The library also defines other information that a container needs to manage deployments for training and inference. After you build a Docker image, you can push it to the Amazon Elastic Container Registry (Amazon ECR). To create a container, you can pull the image from Amazon ECR and build the container using the `docker build` command.
The following high-level schematic shows how the files are organized in an Amazon SageMaker-compatible container created with the Amazon SageMaker Containers library.

When Amazon SageMaker trains a model, it creates a number of files in the container's /opt/ml directory.

```
/opt/ml
### input
#   ### config
#     ### hyperparameters.json
#     ### resourceConfig.json
#   ### data
#       ### <channel_name>
#           ### <input data>
### model
#   ### code
#       ### <script files>
#   ### output
#   ### failure
```

When you run a model training job, the Amazon SageMaker container has a /opt/ml/input/ directory that contains JSON files that configure the hyperparameters for the algorithm and the network layout used for distributed training. The directory also contains files that specify the channels through which Amazon SageMaker accesses the data in Amazon Simple Storage Service (Amazon S3). Place scripts to run in the /opt/ml/code/ directory. The /opt/ml/model/ directory contains the model generated by your algorithm in a single file or an entire directory tree in any format. You can also send information about why a training job failed to the /opt/ml/output/ directory. Amazon SageMaker packages files in this directory into a compressed tar archive file.

When you host a trained model on Amazon SageMaker to make inferences, you deploy the model to an HTTP endpoint. The model makes real-time predictions in response to inference requests. The container must contain a serving stack to process these requests. The five files used in the standard Python serving stack by Amazon SageMaker are installed in the container's WORKDIR. You can choose a different toolset to deploy an HTTP endpoint and, therefore, could have a different layout. If you're writing in a programming language other than Python, you will have a different layout, the nature of which will depend on the frameworks and tools that you choose. The Python serving stack in the WORKDIR directory contains the following files:
• **nginx.conf** – The configuration file for the nginx front end.
• **predictor.py** – The program that implements the Flask web server and the decision tree predictions for this application. You need to customize the code that performs prediction for your application.
• **serve** – The program started when the container is started for hosting. This file simply launches the Gunicorn server, which runs multiple instances of the Flask application defined in predictor.py.
• **train** – The program that is invoked when you run the container for training. To implement your training algorithm, you modify this program.
• **wsgi.py** – A small wrapper used to invoke the Flask application.

In the container, the model files are in the same place that they were written to during training.

```
/opt/ml
### model
### <model files>
```

For more information, see Use Your Own Inference Code (p. 480)

You can provide separate Docker images for the training algorithm and inference code, as shown in the figure. Or you can use a single Docker image for both. When creating Docker images for use with Amazon SageMaker, consider the following:

- Providing two Docker images can increase storage requirements and cost because common libraries might be duplicated.
- In general, smaller containers start faster for both training and hosting. Models train faster and the hosting service can react to increases in traffic by automatically scaling more quickly.
- You might be able to write an inference container that is significantly smaller than the training container. This is especially common when you use GPUs for training, but your inference code is optimized for CPUs.
- Amazon SageMaker requires that Docker containers run without privileged access.
- Docker containers might send messages to the `Stdout` and `Stderr` files. Amazon SageMaker sends these messages to Amazon CloudWatch logs in your AWS account.

### Environmental Variables used by Amazon SageMaker Containers to Define Entry Points

When creating a Dockerfile, you must define an entry point that specifies the location of the code to run when the container starts. Amazon SageMaker Containers does this by setting an `ENV` environment variable. The environment variable that you need to set depends on the job you want to do:

- To run a script, specify the `SAGEMAKER_PROGRAM` `ENV` variable.
- To train an algorithm, specify the `SAGEMAKER_TRAINING_MODULE` `ENV` variable.
- To host a model, specify the `SAGEMAKER_SERVING_MODULE` `ENV` variable.

You can use the Amazon SageMaker containers SDK package to set environment variables.

**SAGEMAKER_PROGRAM**

Train scripts similar to those you would use outside Amazon SageMaker using Amazon SageMaker Script Mode. It supports Python and Shell scripts: Amazon SageMaker uses the Python interpreter for any script with the `.py` suffix. Amazon SageMaker uses the Shell interpreter to execute any other script.

When running a program to specify the entry point for Script Mode, set the `SAGEMAKER_PROGRAM` environmental variable. The script must be located in the `/opt/ml/code` folder.
For example, the container used in the example in Get Started: Use Amazon SageMaker Containers to Run a Python Script (p. 467) sets this \texttt{ENV} as follows.

\begin{verbatim}
ENV SAGEMAKER_PROGRAM train.py
\end{verbatim}

The Amazon SageMaker PyTorch container sets the \texttt{ENV} variable as follows.

\begin{verbatim}
ENV SAGEMAKER_PROGRAM cifar10.py
\end{verbatim}

In the example, \texttt{cifar10.py} is the program that implements the training algorithm and handles loading the model for inferences. For more information, see the \textit{Extending our PyTorch containers} notebook.

\textbf{SAGEMAKER\_TRAINING\_MODULE}

When training an algorithm, specify the location of the module that contains the training logic by setting the \texttt{SAGEMAKER\_TRAINING\_MODULE} environment variable. An Amazon SageMaker container invokes this module when the container starts training. For example, you set this environment variable in MXNet as follows.

\begin{verbatim}
ENV SAGEMAKER\_TRAINING\_MODULE sagemaker\_mxnet\_container\_training:main
\end{verbatim}

For TensorFlow, set this environment variable as follows.

\begin{verbatim}
ENV SAGEMAKER\_TRAINING\_MODULE sagemaker\_tensorflow\_container\_training:main
\end{verbatim}

The code that implements this logic is in Amazon SageMaker Containers.

\textbf{SAGEMAKER\_SERVING\_MODULE}

To locate the module that contains the hosting logic when deploying a model, set the \texttt{SAGEMAKER\_SERVING\_MODULE} environmental variable. An Amazon SageMaker container invokes this module when it starts hosting.

\begin{verbatim}
ENV SAGEMAKER\_SERVING\_MODULE sagemaker\_mxnet\_container\_serving:main
\end{verbatim}

The code that implements this logic is in Amazon SageMaker Containers.

\textbf{Environmental Variables used by Amazon SageMaker Containers Important for Running User Scripts}

When you write a script to run in a container, you are likely to use the following build-time environment variables. Amazon SageMaker Containers sets some of these variable values by default.

- \texttt{SM\_MODEL\_DIR}

\begin{verbatim}
SM\_MODEL\_DIR=/opt/ml/model
\end{verbatim}

When the training job finishes, Amazon SageMaker deletes the container, including its file system, except for the files in the /\texttt{opt/ml/model} and /\texttt{opt/ml/output} folders. Use /\texttt{opt/ml/model} to save the model checkpoints. Amazon SageMaker uploads these checkpoints to the default S3 bucket. Examples:

\begin{verbatim}
# Using it in argparse
parser.add_argument('model_dir', type=str, default=os.environ['SM\_MODEL\_DIR'])

# Using it as a variable
model_dir = os.environ['SM\_MODEL\_DIR']
\end{verbatim}
# Saving checkpoints to the model directory in Chainer
serializers.save_npz(os.path.join(os.environ['SM_MODEL_DIR'], 'model.npz'), model)

For more information, see How Amazon SageMaker Processes Training Output.

• **SM_CHANNELS**

```
SM_CHANNELS=['"testing","training"']
```

The `SM_CHANNELS` environmental variable contains the list of input data channels for the container. When you train a model, you can partition your training data into different logical "channels". Common channels are: training, testing, and evaluation, or images and labels. `SM_CHANNELS` includes the name of the channels that are in the container as a JSON encoded list.

Examples:

```python
import json
# Using it in argparse
parser.add_argument('channel_names', type=int,
                   default=json.loads(os.environ['SM_CHANNELS']))

# Using it as a variable
cchannel_names = json.loads(os.environ['SM_CHANNELS'])
```

• **SM_CHANNEL_{channel_name}**

```
SM_CHANNEL_TRAINING='/opt/ml/input/data/training'
SM_CHANNEL_TESTING='/opt/ml/input/data/testing'
```

The `SM_CHANNEL_{channel_name}` environmental variable contains the directory where the channel named `channel_name` is located in the container.

Examples:

```python
import json
parser.add_argument('--train', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
parser.add_argument('--test', type=str, default=os.environ['SM_CHANNEL_TESTING'])

args = parser.parse_args()
train_file = np.load(os.path.join(args.train, 'train.npz'))
test_file = np.load(os.path.join(args.test, 'test.npz'))
```

• **SM_HPS**

```
SM_HPS='{"batch-size": "256", "learning-rate": "0.0001","communicator": "pure_nccl"}'
```

The `SM_HPS` environmental variable contains a JSON encoded dictionary with the hyperparameters that you have provided.

Example:

```python
import json
hyperparameters = json.loads(os.environ['SM_HPS'])
```
Use Your Own Algorithms or Models

• **SM_HP_ {hyperparameter_name}**

  SM_HP_LEARNING-RATE=0.0001
  SM_HP_BATCH-SIZE=10000
  SM_HP_COMMUNICATOR=pure_nccl

  The SM_HP_ {hyperparameter_name} environmental variable contains the value of the hyperparameter named hyperparameter_name.

  Examples:

  ```python
  learning_rate = float(os.environ['SM_HP_LEARNING-RATE'])
  batch_size = int(os.environ['SM_HP_BATCH-SIZE'])
  comminicator = os.environ['SM_HP_COMMUNICATOR']
  ```

• **SM_CURRENT_HOST**

  SM_CURRENT_HOST=algo-1

  The SM_CURRENT_HOST contains the name of the current container on the container network.

  Examples:

  ```python
  # Using it in argparse
  parser.add_argument('current_host', type=str, default=os.environ['SM_CURRENT_HOST'])
  # Using it as a variable
  current_host = os.environ['SM_CURRENT_HOST']
  ```

• **SM_HOSTS**

  SM_HOSTS='["algo-1","algo-2"]'

  The SM_HOSTS environmental variable contains a JSON-encoded list of all of the hosts.

  Example:

  ```python
  import json
  # Using it in argparse
  parser.add_argument('hosts', type=nargs, default=json.loads(os.environ['SM_HOSTS']))
  # Using it as variable
  hosts = json.loads(os.environ['SM_HOSTS'])
  ```

• **SM_NUM_GPUS**

  SM_NUM_GPUS=1

  The SM_NUM_GPUS environmental variable contains the number of GPUs available in the current container.

  Examples:

  ```python
  # Using it in argparse
  ```
parser.add_argument('num_gpus', type=int, default=os.environ['SM_NUM_GPUS'])

# Using it as a variable
num_gpus = int(os.environ['SM_NUM_GPUS'])

Reference: Amazon SageMaker Containers Environmental Variables

The following build-time environment variables are also defined by default when you use the Amazon SageMaker Containers.

- **SM_NUM_CPUS**

  SM_NUM_CPUs=32

  The `SM_NUM_CPUs` environment variable contains the number of CPUs available in the current container.

  Example:

  ```python
  # Using it in argparse
  parser.add_argument('num_cpus', type=int, default=os.environ['SM_NUM_CPUS'])
  
  # Using it as a variable
  num_cpus = int(os.environ['SM_NUM_CPUS'])
  ```

- **SM_LOG_LEVEL**

  SM_LOG_LEVEL=20

  The `SM_LOG_LEVEL` environment variable contains the current log level in the container.

  Example:

  ```python
  import logging
  
  logger = logging.getLogger(__name__)
  
  logger.setLevel(int(os.environ.get('SM_LOG_LEVEL', logging.INFO)))
  ```

- **SM_NETWORK_INTERFACE_NAME**

  SM_NETWORK_INTERFACE_NAME=ethwe

  The `SM_NETWORK_INTERFACE_NAME` environment variable contains the name of the network interface, which is used for distributed training.

  Example:

  ```python
  # Using it in argparse
  parser.add_argument('network_interface', type=str,
                      default=os.environ['SM_NETWORK_INTERFACE_NAME'])
  
  # Using it as a variable
  network_interface = os.environ['SM_NETWORK_INTERFACE_NAME']
  ```

- **SM_USER_ARGS**
The **SM_INPUT_DIR** environment variable contains a JSON-encoded list of the script arguments provided for training.

- **SM_INPUT_DIR**

```bash
SM_INPUT_DIR=/opt/ml/input/
```

The **SM_INPUT_DIR** environment variable contains the path of the input directory, `/opt/ml/input/`. This is the directory where Amazon SageMaker saves input data and configuration files before and during training.

- **SM_INPUT_CONFIG_DIR**

```bash
SM_INPUT_DIR=/opt/ml/input/config
```

The **SM_INPUT_CONFIG_DIR** environment variable contains the path of the input config directory, `/opt/ml/input/config/`. This is the directory where standard Amazon SageMaker configuration files are located.

When training starts, Amazon SageMaker creates the following files in this directory:

- `hyperparameters.json` – Contains the hyperparameters specified in the `CreateTrainingJob` request.
- `inputdataconfig.json` – Contains the data channel information that you specified in the `InputDataConfig` parameter in a `CreateTrainingJob` request.
- `resourceconfig.json` – Contains the name of the current host and all host containers used in the training.

For more information, see: **Using Your Own Training Algorithms**.

- **SM_OUTPUT_DATA_DIR**

```bash
SM_OUTPUT_DATA_DIR=/opt/ml/output/data/algo-1
```

The **SM_OUTPUT_DATA_DIR** environment variable contains the directory where the algorithm writes non-model training artifacts, such as evaluation results. Amazon SageMaker retains these artifacts. As it runs in a container, your algorithm generates output, including the status of the training job and model, and the output artifacts. Your algorithm should write this information to this directory.

- **SM_RESOURCE_CONFIG**

```bash
SM_RESOURCE_CONFIG='{"current_host":"algo-1","hosts": ["algo-1","algo-2"]}'
```

The **SM_RESOURCE_CONFIG** environment variable contains the contents of the `resourceconfig.json` file located in the `/opt/ml/input/config/` directory. It has the following keys:

- `current_host` – The name of the current container on the container network. For example, "algo-1".
- `hosts` – The list of names of all of the containers on the container network, sorted lexicographically. For example, ["algo-1", "algo-2", "algo-3"] for a three-node cluster.

For more information about the `resourceconfig.json` file, see: **Distributed Training Configuration**.
The `SM_INPUT_DATA_CONFIG` environment variable contains the input data configuration of the `inputdataconfig.json` file located in the `/opt/ml/input/config/` directory.

For more information about the `resourceconfig.json` file, see Distributed Training Configuration.

- **SM_TRAINING_ENV**

```json
SM_TRAINING_ENV="{
    "channel_input_dirs": {
        "test": "/opt/ml/input/data/testing",
        "train": "/opt/ml/input/data/training"
    },
    "current_host": "algo-1",
    "framework_module": "sagemaker_chainer_container.training:main",
    "hosts": ["algo-1", "algo-2"],
    "hyperparameters": {
        "batch-size": 10000,
        "epochs": 1
    },
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "test": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        },
        "train": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        }
    },
    "input_dir": "/opt/ml/input",
    "log_level": 20,
    "model_dir": "/opt/ml/model",
    "module_dir": "s3://sagemaker-{aws-region}-{aws-id}/{training-job-name}/source/sourcedir.tar.gz",
    "module_name": "user_script",
    "network_interface_name": "ethwe",
    "num_gpus": 4,
    "num_gpus": 1,
    "output_data_dir": "/opt/ml/output/data/algo-1",
    "output_dir": "/opt/ml/output",
    "resource_config": {
        "current_host": "algo-1",
```
"hosts": [
    "algo-1",
    "algo-2"
]
}
'

The **SM_TRAINING_ENV** environment variable provides all of the training information as a JSON-encoded dictionary.

**Additional Information for Scripts**

Scripts can assign values for the hyperparameters of an algorithm. The interpreter passes all hyperparameters specified in the training job to the entry point as script arguments. For example, it passes the training job hyperparameters as follow:

```json
{"HyperParameters": {"batch-size": 256, "learning-rate": 0.0001, "communicator": "pure_nccl"}}
```

When an entry point needs additional information from the container that isn't available in hyperparameters, Amazon SageMaker Containers writes this information as environment variables that are available in the script. For example, the following training job includes the training and testing channels.

```python
from sagemaker.pytorch import PyTorch
estimator = PyTorch(entry_point='train.py', ...)
estimator.fit({'training': 's3://bucket/path/to/training/data',
               'testing': 's3://bucket/path/to/testing/data'})
```

The environment variable **SM_CHANNEL_{channel_name}** provides the path where the channel is located.

```python
import argparse
import os
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    ...
    # reads input channels training and testing from the environment variables
    parser.add_argument('--training', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
    parser.add_argument('--testing', type=str, default=os.environ['SM_CHANNEL_TESTING'])
    args = parser.parse_args()
    ...
```

**Get Started: Use Amazon SageMaker Containers to Run a Python Script**

To run an arbitrary script-based program in a Docker container using the Amazon SageMaker Containers, build a Docker container with an Amazon SageMaker notebook instance, as follows:

1. Create the notebook instance.
2. Create and upload the Dockerfile and Python scripts.
3. Build the container.
4. Test the container locally.
5. Clean up the resources.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebook, Notebook instances, and Create notebook instance.
3. On the Create notebook instance page, provide the following information:
   a. For Notebook instance name, enter RunScriptNotebookInstance.
   b. For Notebook Instance type, choose ml.t2.medium.
   c. For IAM role, choose Create a new role.
      i. Choose Create a new role.
      ii. On the Create an IAM role page, choose Specific S3 buckets, specify an S3 bucket named sagemaker-run-script, and then choose Create role.
      Amazon SageMaker creates an IAM role named AmazonSageMaker-ExecutionRole-YYYYMMDDHHmmsS. For example, AmazonSageMaker-ExecutionRole-20190429T110788. Record the role name because you'll need it later.
   d. For Root Access, accept the default, Enabled.
   e. Choose Create notebook instance.

   It takes a few minutes for Amazon SageMaker to launch an ML compute instance—in this case, a notebook instance—and attach an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the CreateNotebookInstance (p. 913) API.
4. When the status of the notebook instance is InService, from Actions, choose Open Jupyter.
   For New, choose conda_tensorflow_p36. This is the kernel you need.
5. To name the notebook, choose File, Rename, enter Run-Python-Script, and then choose Rename.

To create and upload the Dockerfile and Python scripts

1. In the editor of your choice, create the following Dockerfile text file locally and save it with the file name "Dockerfile" without an extension. The docker build command expects by default to find a file with precisely this name in the dockerfile directory. For example, in Notepad, you can save a text file without an extension by choosing File, Save As and choosing all types(*.*)

```
FROM tensorflow/tensorflow:2.0.0a0

RUN pip install sagemaker-containers

# Copies the training code inside the container
COPY train.py /opt/ml/code/train.py

# Defines train.py as script entrypoint
ENV SAGEMAKER_PROGRAM train.py
```

The Dockerfile script performs the following tasks:
• **FROM** tensorflow/tensorflow:2.0.0a0 downloads the TensorFlow library used to run the Python script.
• **RUN** `pip install sagemaker-containers` Amazon SageMaker Containers contains the common functionality necessary to create a container compatible with Amazon SageMaker.
• **COPY** `train.py /opt/ml/code/train.py` copies the script to the location inside the container that is expected by Amazon SageMaker. The script must be located in this folder.
• **ENV SAGEMAKER_PROGRAM** `train.py` defines `train.py` as the name of the entrypoint script that is located in the /opt/ml/code folder for the container. This is the only environmental variable that you must specify when, you are using your own container.

2. In the editor of your choice, create and save the following `train.py` text file locally.

```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                      tf.keras.layers.Dense(128, activation='relu'),
                                      tf.keras.layers.Dropout(0.2),
                                      tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

model.fit(x_train, y_train, epochs=1)
model.evaluate(x_test, y_test)
```

3. To upload the Dockerfile to a dockerfile directory, choose Open JupyterLab, choose the File Browser icon, and then choose the New Folder icon. This creates a new directory named dockerfile.
4. Double-click the new dockerfile folder, choose the Upload Files icon, navigate to where you saved your Dockerfile and train.py script files, and upload them to the dockerfile folder.

**To build the container**

1. The Jupyter Notebook opens in the SageMaker directory. The Docker build command must be run from the dockerfile directory you created. Run the following command to change into the dockerfile directory:

```
  cd dockerfile
```

This returns your current directory: /home/ec2-user/SageMaker/dockerfile

2. To build the Docker container, run the following Docker build command, including the final period.

```
!docker build -t tf-2.0 .
```

**To test the container locally**

1. Use Local Mode the test the container locally. Replace the 'SageMakerRole' value with the ARN for the role with the IAM role you created when configuring the notebook instance. The ARN
should look like: 

```python
from sagemaker.estimator import Estimator
estimator = Estimator(image_name='tf-2.0',
                      role='SageMakerRole',
                      train_instance_count=1,
                      train_instance_type='local')
estimator.fit()
```

This test outputs the training environment configuration, the values used for the environmental variables, the source of the data, and the loss and accuracy obtained during training.

2. After using Local Mode, you can push the image to Amazon Elastic Container Registry and use it to run training jobs. For an example that shows how to complete these tasks, see Building Your Own TensorFlow Container

To clean up resources when done with the get started example

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/, stop and then delete the notebook instance.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3 and delete the bucket that you created for storing model artifacts and the training dataset.
3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.

   **Note**
   The Docker container shuts down automatically after it has run. You don’t need to delete it.

**Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch**

Amazon SageMaker provides prebuilt Docker images that include deep learning framework libraries and other dependencies needed for training and inference. With the SageMaker Python SDK, you can train and deploy models using one of these popular deep learning frameworks. For instructions on installing and using the SDK, see Amazon SageMaker Python SDK.

The following table provides links to the GitHub repositories that contain the source code and Dockerfiles for each framework and for TensorFlow and MXNet Serving. The instructions linked are for using the Python SDK estimators to run your own training algorithms on Amazon SageMaker and your own models on Amazon SageMaker hosting.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Prebuilt Docker Image Source Code</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TensorFlow</strong></td>
<td><a href="#">Amazon SageMaker TensorFlow Containers</a></td>
<td>Using TensorFlow with the SageMaker Python SDK</td>
</tr>
<tr>
<td></td>
<td><a href="#">Amazon SageMaker TensorFlow Serving Container</a></td>
<td></td>
</tr>
<tr>
<td><strong>MXNet</strong></td>
<td><a href="#">Amazon SageMaker MXNet Containers</a></td>
<td>Using MXNet with the SageMaker Python SDK</td>
</tr>
<tr>
<td></td>
<td><a href="#">Amazon SageMaker MXNet Serving Container</a></td>
<td></td>
</tr>
</tbody>
</table>
If you are not using the Amazon SageMaker Python SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-built container. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). To pull an image from an Amazon ECR repo or to push an image to an Amazon ECR repo, use the fullname registry address of the image. Amazon SageMaker uses the following URL patterns for the container image registry addresses:

```
<account_id>.dkr.ecr.<region>.amazonaws.com/<ECR repo name>:<framework version>-<processing unit type>-<python version>
```

Use the following commands to pull these images:

```
$(aws ecr get-login --no-include-email --registry-ids <account_id>)
docker pull <account_id>.dkr.ecr.<region>.amazonaws.com/<ECR repo name>:<framework version>-<processing unit type>-<python version>
```

The following table itemizes the supported values for each of the components in the URL registry addresses and how they are associated.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Prebuilt Docker Image Source Code</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chainer</td>
<td>Amazon SageMaker Chainer SageMaker Containers</td>
<td>Chainer SageMaker Estimators and Models</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Amazon SageMaker PyTorch Containers</td>
<td>SageMaker PyTorch Estimators and Models</td>
</tr>
</tbody>
</table>

Here, for example, are some of the most common use cases for the deep learning frameworks supported the Amazon SageMaker:

- If you want to use **TensorFlow 1.13** or later to **train** a model:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-training:1.13-gpu

- If you want to use **TensorFlow 1.14** or later to **train** a model:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-training:1.14-gpu

- If you want to use **TensorFlow 1.14** or later for **inference**:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-inference:1.14-gpu

- If you want to use **MxNet 1.4.1** or later to **train** a model or for **inference** with Python 3:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/mxnet-training:1.4.1-gpu-py3
  
  763104351884.dkr.ecr.<region>.amazonaws.com/mxnet-inference:1.4.1-gpu-py3

- If you want to use **PyTorch 1.2.0** or later to **train** a model or for **inference** with Python 3:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/pytorch-training:1.2.0-gpu-py3
  
  763104351884.dkr.ecr.<region>.amazonaws.com/pytorch-inference:1.2.0-gpu-py3

- If you want to use **Chainer** to **train** a model or for **inference** with Python 2 or 3:
  
  520713654638.dkr.ecr.<region>.amazonaws.com/sagemaker-chainer:5.0.0-gpu-<python version>

- If you want to use **EIA** containers for **inference** with **TensorFlow 1.14** with Python 3 or for **MxNet 1.4.1** with Python 2 or 3:
  
  763104351884.dkr.ecr.<region>.amazonaws.com/tensorflow-inference-eia:1.14-cpu
<table>
<thead>
<tr>
<th>URL Component</th>
<th>Description</th>
<th>Supported Values</th>
</tr>
</thead>
</table>
| <account_id>  | Specifies the ID for Amazon SageMaker accounts that contain the pre-built containers. | • 763104351884  
• 520713654638  
• 871362719292  
• 057415533634 |
| <region>      | Specifies the AWS regions that contain the Amazon SageMaker accounts. | • Accounts 763104351884 and 520713654638 are located in:  
us-west-1, us-west-2, us-east-1, us-east-2,  
ap-northeast-1, ap-northeast-2, ap-southeast-1, ap-southeast-2, ap-south-1,  
eu-west-1, eu-west-2, eu-central-1, ca-central-1  
• Accounts 871362719292 and 057415533634 are located in:  
ap-east-1  
• Account 763104351884 has EIA containers located in:  
us-west-2, us-east-1, us-east-2, eu-west-1,  
ap-northeast-1, ap-northeast-2 |
| <ECR repo name> | Specifies the name of the public repository owned by Amazon SageMaker in the Amazon ECR. | Python 3 containers for TensorFlow-1.13 and later, for MXNet-1.4.1, and for PyTorch 1.2.0 and later in accounts 763104351884 and 871362719292:  
• tensorflow-training (also for Python 2)  
tensorflow-inference  
mxnet-training  
mxnet-inference  
pytorch-training (also for Python 2)  
pytorch-inference (also for Python 2)  
Other Python 2 and Python 3 containers in accounts 520713654638 and 057415533634:  
sagemaker-tensorflow-scriptmode (for Python 2 only)  
sagemaker-tensorflow-serving-eia  
sagemaker-mxnet (for Python 2 only)  
sagemaker-mxnet-serving (for Python 2 only)  
sagemaker-mxnet-serving-eia  
sagemaker-chainer  
sagemaker-pytorch |
Amazon SageMaker Developer Guide
Use Your Own Algorithms or Models

<table>
<thead>
<tr>
<th>URL Component</th>
<th>Description</th>
<th>Supported Values</th>
</tr>
</thead>
</table>
| <framework version> | Specifies the framework and links to documentation for the estimators for each of the frameworks that explains how to specify the supported versions. | • TensorFlow: TensorFlow SageMaker Estimators  
• MXNet: MXNet SageMaker Estimators  
• Chainer: Chainer SageMaker Estimators  
• PyTorch: PyTorch SageMaker Estimators |
| <processing unit type> | Specifies whether to use a GPU or CPU for training or hosting. | • cpu (EIA images also use cpu in the label.)  
• gpu |
| <python version> | Specifies the version of Python used. (This tag is not used for the tensorflow-inference, tensorflow-inference-eia, and tensorflow-serving containers.) | • py2  
• py3 |

Amazon SageMaker also provides prebuilt Docker images for scikit-learn and Spark ML. For information about Docker images that enable using scikit-learn and Spark ML solutions in Amazon SageMaker, see Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML (p. 473).

You can use prebuilt containers to deploy your custom models or models that you have purchased on AWS Marketplace that have been trained in a framework other than Amazon SageMaker. For an overview of the process of bringing the trained model artifacts into Amazon SageMaker and hosting them at an endpoint, see Bring Your Own Pretrained MXNet or TensorFlow Models into Amazon SageMaker.

You can customize these prebuilt containers or extend them to handle any additional functional requirements for your algorithm or model that the prebuilt Amazon SageMaker Docker image doesn't support. For an example, see Extending Our PyTorch Containers.

**Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML**

Amazon SageMaker provides prebuilt Docker images that install the scikit-learn and Spark ML libraries and the dependencies they need to build Docker images that are compatible with Amazon SageMaker using the Amazon SageMaker Python SDK. With the SDK, you can use scikit-learn for machine learning tasks and use Spark ML to create and tune machine learning pipelines. For instructions on installing and using the SDK, see SageMaker Python SDK. The following table contains links to the GitHub repositories with the source code and the Dockerfiles for scikit-learn and Spark ML frameworks and to instructions that show how use the Python SDK estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker Hosting.
If you are not using the SM Python SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-build container. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). You can push or pull them using their fullname registry addresses. Amazon SageMaker uses the following Docker Image URL patterns for scikit-learn and Spark M:

- `<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-scikit-learn`
  
  For example, 746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-scikit-learn

- `<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-sparkml-serving`
  
  For example, 341280168497.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-sparkml-serving

The following table lists the supported values for account IDs and corresponding AWS Region names.

<table>
<thead>
<tr>
<th>ACCOUNT_ID</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>746614075791</td>
<td>us-west-1</td>
</tr>
<tr>
<td>246618743249</td>
<td>us-west-2</td>
</tr>
<tr>
<td>683313688378</td>
<td>us-east-1</td>
</tr>
<tr>
<td>257758044811</td>
<td>us-east-2</td>
</tr>
<tr>
<td>354813040037</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>366743142698</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>121021644041</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>783357654285</td>
<td>ap-southeast-2</td>
</tr>
<tr>
<td>720646828776</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>141502667606</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>764974769150</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>492215442770</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>341280168497</td>
<td>ca-central-1</td>
</tr>
<tr>
<td>414596584902</td>
<td>us-gov-west-1</td>
</tr>
</tbody>
</table>
Amazon SageMaker also provides prebuilt Docker images for popular deep learning frameworks. For information about Docker images that enable using deep learning frameworks in Amazon SageMaker, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 470).

For information on Docker images for developing reinforcement learning (RL) solutions in Amazon SageMaker, see Amazon SageMaker RL Containers.

Example Notebooks: Use Your Own Algorithm or Model

The following sample notebooks show how to use your own algorithms or pretrained models from an Amazon SageMaker notebook instance. After you have created a notebook instance and opened it, choose the SageMaker Examples tab for a list of all Amazon SageMaker example notebooks. You can open the sample notebooks from the Advanced Functionality section in your notebook instance or in GitHub at the provided links. To open a notebook, choose its Use tab, then choose Create copy.

For instructions on how to create and access Jupyter notebook instances, see Use Amazon SageMaker Notebook Instances (p. 201)

To learn how to host models trained in scikit-learn for making predictions in Amazon SageMaker by injecting them first-party k-means and XGBoost containers, see the following sample notebooks.

- kmeans_bring_your_own_model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/kmeans_bring_your_own_model
- xgboost_bring_your_own_model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/xgboost_bring_your_own_model

To learn how to package algorithms that you have developed in TensorFlow and scikit-learn frameworks for training and deployment in the Amazon SageMaker environment, see the following notebooks. They show you how to build, register, and deploy your own Docker containers using Dockerfiles.

- tensorflow_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/tensorflow_bring_your_own
- scikit_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/scikit_bring_your_own

To learn how to train a neural network locally using MXNet or TensorFlow, and then create an endpoint from the trained model and deploy it on Amazon SageMaker, see the following notebooks. The MXNet model is trained to recognize handwritten numbers from the MNIST dataset. The TensorFlow model is trained to classify irises.


To learn how to use a Dockerfile to build a container that calls the train.py script and uses pipe mode to custom train an algorithm, see the following notebook. In pipe mode, the input data is transferred to the algorithm while it is training. This can decrease training time compared to using file-mode.

To learn how to use an R container to train and host a model with the R kernel installed in a notebook, see the following notebook. To take advantage of the AWS SDK for Python (Boto 3), we use Python within the notebook. You can achieve the same results completely in R by invoking command line arguments.

- **rBringYourOwn** - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/r_bring_your_own

To learn how to extend a prebuilt Amazon SageMaker PyTorch container image when you have additional functional requirements for your algorithm or model that the pre-built Docker image doesn't support, see the following notebook.

- **pytorchExtendingOurContainers** - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/pytorch_extending_our_containers

For links to the GitHub repositories with the prebuilt Dockerfiles for the TensorFlow, MXNet, Chainer, and PyTorch frameworks and instructions on use the AWS SDK for Python (Boto 3) estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker hosting, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 470)

### Use Your Own Training Algorithms

This section explains how Amazon SageMaker interacts with a Docker container that runs your custom training algorithm. Use this information to write training code and create a Docker image for your training algorithms.

**Topics**

- How Amazon SageMaker Runs Your Training Image (p. 476)
- How Amazon SageMaker Provides Training Information (p. 477)
- How Amazon SageMaker Signals Algorithm Success and Failure (p. 480)
- How Amazon SageMaker Processes Training Output (p. 480)

### How Amazon SageMaker Runs Your Training Image

To configure a Docker container to run as an executable, use an ENTRYPOINT instruction in a Dockerfile. Note the following:

- For model training, Amazon SageMaker runs the container as follows:

  ```bash
docker run image train
  ```

  Amazon SageMaker overrides any default CMD statement in a container by specifying the train argument after the image name. The train argument also overrides arguments that you provide using CMD in the Dockerfile.

- Use the exec form of the ENTRYPOINT instruction:

  ```bash
  ENTRYPOINT ["executable", "param1", "param2", ...]
  ```

  For example:
The `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from Amazon SageMaker APIs. Note the following:

- The `CreateTrainingJob (p. 931)` API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.

- The `StopTrainingJob (p. 1224)` API issues the equivalent of the `docker stop`, with a 2 minute timeout, command to gracefully stop the specified container:

```
docker stop -t120
```

The command attempts to stop the running container by sending a `SIGTERM` signal. After the 2 minute timeout, `SIGKILL` is sent and the containers are forcibly stopped. If the container handles the `SIGTERM` gracefully and exits within 120 seconds from receiving it, no `SIGKILL` is sent.

**Note**

If you want access to the intermediate model artifacts after Amazon SageMaker stops the training, add code to handle saving artifacts in your `SIGTERM` handler.

- If you plan to use GPU devices for model training, make sure that your containers are `nvidia-docker` compatible. Only the CUDA toolkit should be included on containers; don't bundle NVIDIA drivers with the image. For more information about `nvidia-docker`, see [NVIDIA/nvidia-docker](https://github.com/nvidia/nvidia-docker).

- You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the `train` and `serve` arguments.

- `/opt/ml` and all sub-directories are reserved by Amazon SageMaker training. When building your algorithm's docker image, please ensure you don't place any data required by your algorithm under them as the data may no longer be visible during training.

**How Amazon SageMaker Provides Training Information**

This section explains how Amazon SageMaker makes training information, such as training data, hyperparameters, and other configuration information, available to your Docker container.

When you send a `CreateTrainingJob (p. 931)` request to Amazon SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contains the training algorithm. You also specify the Amazon Simple Storage Service (Amazon S3) location where training data is stored and algorithm-specific parameters. Amazon SageMaker makes this information available to the Docker container so that your training algorithm can use it. This section explains how we make this information available to your Docker container. For information about creating a training job, see `CreateTrainingJob`.

**Topics**

- Hyperparameters (p. 478)
- Environment Variables (p. 478)
- Input Data Configuration (p. 478)
- Training Data (p. 478)
- Distributed Training Configuration (p. 479)
Hyperparameters

Amazon SageMaker makes the hyperparameters in a CreateTrainingJob request available in the Docker container in the /opt/ml/input/config/hyperparameters.json file.

Environment Variables

- TRAINING_JOB_NAME—The training job name stored in the TrainingJobName parameter in a CreateTrainingJob (p. 931) request.
- TRAINING_JOB_ARN—The Amazon Resource Name (ARN) of the training job returned as the TrainingJobArn response element for CreateTrainingJob (p. 931).

Input Data Configuration

You specify data channel information in the InputDataConfig parameter in a CreateTrainingJob request. Amazon SageMaker makes this information available in the /opt/ml/input/config/inputdataconfig.json file in the Docker container.

For example, suppose that you specify three data channels (train, evaluation, and validation) in your request. Amazon SageMaker provides the following JSON:

```json
{
    "train" : {
        "ContentType": "trainingContentType",
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"},
    "evaluation" : {
        "ContentType": "evalContentType",
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"},
    "validation" : {
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"}
}
```

Note

Amazon SageMaker provides only relevant information about each data channel (for example, the channel name and the content type) to the container, as shown. S3DistributionType will be set as FullyReplicated if specify EFS or FSxLustre as input data sources.

Training Data

The TrainingInputMode parameter in a CreateTrainingJob request specifies how to make data available for model training: in FILE mode or PIPE mode. Depending on the specified input mode, Amazon SageMaker does the following:

- FILE mode—Amazon SageMaker makes the data for the channel available in the /opt/ml/input/data/channel_name directory in the Docker container. For example, if you have three channels named training, validation, and testing, Amazon SageMaker makes three directories in the Docker container:
  - /opt/ml/input/data/training
  - /opt/ml/input/data/validation
  - /opt/ml/input/data/testing

  Note
  Channels that use file system data sources such as Amazon Elastic File System (EFS) and Amazon FSx must use FILE mode. Also to utilize an FSx file server, you must specify a
path that begins with /fsx. If a file system is specified, the directory path provided in the channel is mounted at /opt/ml/input/data/channel_name.

- PIPE mode—Amazon SageMaker makes data for the channel available from the named pipe: /opt/ml/input/data/channel_name_epoch_number. For example, if you have three channels named training, validation, and testing, you will need to read from the following pipes:
  - /opt/ml/input/data/training_0, /opt/ml/input/data/training_1, ...
  - /opt/ml/input/data/validation_0, /opt/ml/input/data/validation_1, ...
  - /opt/ml/input/data/testing_0, /opt/ml/input/data/testing_1, ...

Read the pipes sequentially. For example, if you have a channel called training, read the pipes in this sequence:

1. Open /opt/ml/input/data/training_0 in read mode and read it to end-of-file (EOF), or if you are done with the first epoch, close the pipe file early.
2. After closing the first pipe file, look for /opt/ml/input/data/training_1 and read it until you have completed the second epoch, and so on.

If the file for a given epoch doesn’t exist yet, your code may need to retry until the pipe is created. There is no sequencing restriction across channel types. That is, you can read multiple epochs for the training channel, for example, and only start reading the validation channel when you are ready. Or, you can read them simultaneously if your algorithm requires that.

Distributed Training Configuration

If you're performing distributed training with multiple containers, Amazon SageMaker makes information about all containers available in the /opt/ml/input/config/resourceconfig.json file.

To enable inter-container communication, this JSON file contains information for all containers. Amazon SageMaker makes this file available for both FILE and PIPE mode algorithms. The file provides the following information:

- current_host—The name of the current container on the container network. For example, algo-1. Host values can change at any time. Don't write code with specific values for this variable.
- hosts—The list of names of all containers on the container network, sorted lexicographically. For example, ["algo-1", "algo-2", "algo-3"] for a three-node cluster. Containers can use these names to address other containers on the container network. Host values can change at any time. Don't write code with specific values for these variables.
- network_interface_name—The name of the network interface that is exposed to your container. For example, containers running the Message Passing Interface (MPI) can use this information to set the network interface name.
- Do not use the information in /etc/hostname or /etc/hosts because it might be inaccurate.
- Hostname information may not be immediately available to the algorithm container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

The following is an example file on node 1 in a three-node cluster:

```json
{
"current_host": "algo-1",
"hosts": ["algo-1","algo-2","algo-3"],
"network_interface_name": "eth1"
}
```
How Amazon SageMaker Signals Algorithm Success and Failure

A training algorithm indicates whether it succeeded or failed using the exit code of its process.

A successful training execution should exit with an exit code of 0 and an unsuccessful training execution should exit with a non-zero exit code. These will be converted to "Completed" and "Failed" in the TrainingJobStatus returned by DescribeTrainingJob. This exit code convention is standard and is easily implemented in all languages. For example, in Python, you can use `sys.exit(1)` to signal a failure exit and simply running to the end of the main routine will cause Python to exit with code 0.

In the case of failure, the algorithm can write a description of the failure to the failure file. See next section for details.

How Amazon SageMaker Processes Training Output

As your algorithm runs in a container, it generates output including the status of the training job and model and output artifacts. Your algorithm should write this information to the following files, which are located in the container's /output directory. Amazon SageMaker processes the information contained in this directory as follows:

- **/opt/ml/output/failure**—If training fails, after all algorithm output (for example, logging) completes, your algorithm should write the failure description to this file. In a DescribeTrainingJob response, Amazon SageMaker returns the first 1024 characters from this file as FailureReason.

- **/opt/ml/model**—Your algorithm should write all final model artifacts to this directory. Amazon SageMaker copies this data as a single object in compressed tar format to the S3 location that you specified in the CreateTrainingJob request. If multiple containers in a single training job write to this directory they should ensure no file/directory names clash. Amazon SageMaker aggregates the result in a tar file and uploads to s3.

Use Your Own Inference Code

You can use Amazon SageMaker to interact with Docker containers and run your own inference code in one of two ways:

- To use your own inference code with a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To use your own inference code to get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

- Use Your Own Inference Code with Hosting Services (p. 480)
- Use Your Own Inference Code with Batch Transform (p. 483)

Use Your Own Inference Code with Hosting Services

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for hosting services. Use this information to write inference code and create a Docker image.

Topics

- How Amazon SageMaker Runs Your Inference Image (p. 481)
How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model inference, Amazon SageMaker runs the container as:

```
docker run image serve
```

Amazon SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the image name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

For example:

```
ENTRYPOINT ["python", "k_means_inference.py"]
```

The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from the Amazon SageMaker APIs, which is a requirement.

For example, when you use the `CreateEndpoint (p. 875)` API to create an endpoint, Amazon SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. Amazon SageMaker runs the Docker container on those instances.

If you reduce the number of instances backing the endpoint (by calling the `UpdateEndpointWeightsAndCapacities (p. 1235)` APIs), Amazon SageMaker runs a command to stop the Docker container on the instances being terminated. The command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

If you update the endpoint (by calling the `UpdateEndpoint (p. 1233)` API), Amazon SageMaker launches another set of ML compute instances and runs the Docker containers that contain your inference code on them. Then it runs a command to stop the previous Docker containers. To stop a Docker container, command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

- Amazon SageMaker uses the container definition that you provided in your `CreateModel (p. 902)` request to set environment variables and the DNS hostname for the container as follows:
• It sets environment variables using the `ContainerDefinition.Environment` string-to-string map.
• It sets the DNS hostname using the `ContainerDefinition.ContainerHostname`.

• If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateEndpointConfig` request), make sure that your containers are `nvidia-docker` compatible. Don't bundle NVIDIA drivers with the image. For more information about `nvidia-docker`, see NVIDIA/nvidia-docker.

• You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

How Amazon SageMaker Loads Your Model Artifacts

In your `CreateModel` request, the container definition includes the `ModelDataUrl` parameter, which identifies the S3 location where model artifacts are stored. Amazon SageMaker uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory for use by your inference code.

The `ModelDataUrl` must point to a tar.gz file. Otherwise, Amazon SageMaker won't download the file.

If you trained your model in Amazon SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside Amazon SageMaker, you need to create this single compressed tar file and save it in a S3 location. Amazon SageMaker decompresses this tar file into `/opt/ml/model` directory before your container starts.

How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

How Your Container Should Respond to Inference Requests

To obtain inferences, the client application sends a POST request to the Amazon SageMaker endpoint. For more information, see the `InvokeEndpoint` API. Amazon SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

• Amazon SageMaker strips all POST headers except those supported by `InvokeEndpoint`. Amazon SageMaker might add additional headers. Inference containers must be able to safely ignore these additional headers.
• To receive inference requests, the container must have a web server listening on port 8080 and must accept POST requests to the `/invocations` endpoint.
• A customer's model containers must accept socket connection requests within 250 ms.
• A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the `/invocations`. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

How Your Container Should Respond to Health Check (Ping) Requests

The `CreateEndpoint` and `UpdateEndpoint` API calls result in Amazon SageMaker starting new inference containers. Soon after container startup, Amazon SageMaker starts sending periodic GET requests to the `/ping` endpoint.
The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

If the container does not begin to pass health checks, by consistently responding with 200s, during the 4 minutes after startup, CreateEndpoint will fail, leaving Endpoint in a failed state, and the update requested by UpdateEndpoint will not be completed.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

### Use Your Own Inference Code with Batch Transform

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for batch transform. Use this information to write inference code and create a Docker image.

**Topics**

- How Amazon SageMaker Runs Your Inference Image (p. 483)
- How Amazon SageMaker Loads Your Model Artifacts (p. 484)
- How Containers Serve Requests (p. 484)
- How Your Container Should Respond to Health Check (Ping) Requests (p. 485)

### How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an ENTRYPOINT instruction in a Dockerfile. Note the following:

- For batch transforms, Amazon SageMaker runs the container as:

  ```bash
  docker run image serve
  ```

  Amazon SageMaker overrides default CMD statements in a container by specifying the serve argument after the image name. The serve argument overrides arguments that you provide with the CMD command in the Dockerfile.

- We recommend that you use the exec form of the ENTRYPOINT instruction:

  ```bash
  ENTRYPOINT ["executable", "param1", "param2"]
  
  For example:
  
  ```bash
  ENTRYPOINT ["python", "k_means_inference.py"]
  ```

- Amazon SageMaker sets environment variables specified in CreateModel (p. 902) and CreateTransformJob (p. 939) on your container. Additionally, the following environment variables will be populated:
  - SAGEMAKER_BATCH is always set to true when the container runs in Batch Transform.
  - SAGEMAKER_MAX_PAYLOAD_IN_MB is set to the largest size payload that will be sent to the container via HTTP.
• **SAGEMAKER_BATCH_STRATEGY** will be set to **SINGLE_RECORD** when the container will be sent a single record per call to invocations and **MULTI_RECORD** when the container will get as many records as will fit in the payload.

• **SAGEMAKER_MAX_CONCURRENT_TRANSFORMS** is set to the maximum number of /invocations requests that can be opened simultaneously.

  **Note**
  The last three environment variables come from the API call made by the user. If the user doesn't set values for them, they aren't passed. In that case, either the default values or the values requested by the algorithm (in response to the /execution-parameters) are used.

• If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your CreateTransformJob request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see NVIDIA/nvidia-docker.

• You can't use the **init** initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

**How Amazon SageMaker Loads Your Model Artifacts**

In a **CreateModel** request, container definitions includes the **ModelDataUrl** parameter, which identifies the location in Amazon S3 where model artifacts are stored. When you use Amazon SageMaker to run inferences, it uses this information to determine where to copy the model artifacts from. It copies the artifacts to the /opt/ml/model directory in the Docker container for use by your inference code.

The **ModelDataUrl** parameter must point to a tar.gz file. Otherwise, Amazon SageMaker can't download the file. If you train a model in Amazon SageMaker, it saves the artifacts as a single compressed tar file in Amazon S3. If you train a model in another framework, you need to store the model artifacts in Amazon S3 as a compressed tar file. Amazon SageMaker decompresses this tar file and saves it in the /opt/ml/model directory in the container before the batch transform job starts.

**How Containers Serve Requests**

Containers must implement a web server that responds to invocations and ping requests on port 8080. For batch transforms, you have the option to set algorithms to implement execution-parameters requests to provide a dynamic runtime configuration to Amazon SageMaker. Amazon SageMaker uses the following endpoints:

• **ping**—Used to periodically check the health of the container. Amazon SageMaker waits for an HTTP 200 status code and an empty body for a successful ping request before sending an invocations request. You might use a ping request to load a model into memory to generate inference when invocations requests are sent.

• (Optional) **execution-parameters**—Allows the algorithm to provide the optimal tuning parameters for a job during runtime. Based on the memory and CPUs available for a container, the algorithm chooses the appropriate MaxConcurrentTransforms, BatchStrategy, and MaxPayloadInMB values for the job.

Before calling the invocations request, Amazon SageMaker attempts to invoke the execution-parameters request. When you create a batch transform job, you can provide values for the MaxConcurrentTransforms, BatchStrategy, and MaxPayloadInMB parameters. Amazon SageMaker determines the values for these parameters using this order of precedence:

1. The parameter values that you provide when you create the CreateTransformJob request,
2. The values that the model container returns when Amazon SageMaker invokes the execution-parameters endpoint
3. The parameters default values, listed in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxConcurrentTransforms</td>
<td>1</td>
</tr>
<tr>
<td>BatchStrategy</td>
<td>MULTI_RECORD</td>
</tr>
<tr>
<td>MaxPayloadInMB</td>
<td>6</td>
</tr>
</tbody>
</table>

The response for a GET execution-parameters request is a JSON object with keys for `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. This is an example of a valid response:

```json
{
  "MaxConcurrentTransforms": 8,
  "BatchStrategy": "MULTI_RECORD",
  "MaxPayloadInMB": 6
}
```

**How Your Container Should Respond to Health Check (Ping) Requests**

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the `/invocations` endpoint.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on `/ping` attempts is 2 seconds.

**Create Algorithm and Model Package Resources**

After your training and/or inference code is packaged in Docker containers, create algorithm and model package resources that you can use in your Amazon SageMaker account and, optionally, publish on AWS Marketplace.

**Topics**

- Create an Algorithm Resource (p. 485)
- Create a Model Package Resource (p. 489)

**Create an Algorithm Resource**

To create an algorithm resource that you can use to run training jobs in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker containers that contains the training and, optionally, inference code.
- The configuration of the input data that your algorithm expects for training.
- The hyperparameters that your algorithm supports.
- Metrics that your algorithm sends to Amazon CloudWatch during training jobs.
- The instance types that your algorithm supports for training and inference, and whether it supports distributed training across multiple instances.
- Validation profiles, which are training jobs that Amazon SageMaker uses to test your algorithm's training code and batch transform jobs that Amazon SageMaker runs to test your algorithm's inference code.
To ensure that buyers and sellers can be confident that products work in Amazon SageMaker, we require that you validate your algorithms before listing them on AWS Marketplace. You can list products in the AWS Marketplace only if validation succeeds. To validate your algorithms, Amazon SageMaker uses your validation profile and sample data to run the following validations tasks:

1. Create a training job in your account to verify that your training image works with Amazon SageMaker.
2. If you included inference code in your algorithm, create a model in your account using the algorithm’s inference image and the model artifacts produced by the training job.
3. If you included inference code in your algorithm, create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

When you list your product on AWS Marketplace, the inputs and outputs of this validation process persist as part of your product and are made available to your buyers. This helps buyers understand and evaluate the product before they buy it. For example, buyers can inspect the input data that you used, the outputs generated, and the logs and metrics emitted by your code. The more comprehensive your validation specification, the easier it is for customers to evaluate your product.

**Note**
In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the Training jobs and Transform jobs pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. If any issues are found, you will have to create the algorithm again.

**Note**
To publish your algorithm on AWS Marketplace, at least one validation profile is required.

You can create an algorithm by using either the Amazon SageMaker console or the Amazon SageMaker API.

**Topics**
- Create an Algorithm Resource (Console) (p. 486)
- Create an Algorithm Resource (API) (p. 489)

**Create an Algorithm Resource (Console)**

**To create an algorithm resource (console)**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms, then choose Create algorithm.
3. On the **Training specifications** page, provide the following information:
   a. **For Algorithm name**, type a name for your algorithm. The algorithm name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   b. **Type a description for your algorithm.** This description appears in the Amazon SageMaker console and in the AWS Marketplace.
   c. **For Training image**, type the path in Amazon ECR where your training container is stored.
   d. **For Support distributed training**, Choose Yes if your algorithm supports training on multiple instances. Otherwise, choose No.
   e. **For Support instance types for training**, choose the instance types that your algorithm supports.
f. For **Channel specification**, specify up to 8 channels of input data for your algorithm. For example, you might specify 3 input channels named `train`, `validation`, and `test`. For each channel, specify the following information:

   i. For **Channel name**, type a name for the channel. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   
   ii. To require the channel for your algorithm, choose **Channel required**.
   
   iii. Type a description for the channel.
   
   iv. For **Supported input modes**, choose **Pipe mode** if your algorithm supports streaming the input data, and **File mode** if your algorithm supports downloading the input data as a file. You can choose both.
   
   v. For **Supported content types**, type the MIME type that your algorithm expects for input data.
   
   vi. For **Supported compression type**, choose **Gzip** if your algorithm supports Gzip compression. Otherwise, choose **None**.
   
   vii. Choose **Add channel** to add another data input channel, or choose **Next** if you are done adding channels.

4. On the **Tuning specifications** page, provide the following information:

   a. For **Hyperparameter specification**, specify the hyperparameters that your algorithm supports by editing the JSON object. For each hyperparameter that your algorithm supports, construct a JSON block similar to the following:

   ```json
   {
     "DefaultValue": "5",
     "Description": "The first hyperparameter",
     "IsRequired": true,
     "IsTunable": false,
     "Name": "intRange",
     "Range": {
       "IntegerParameterRangeSpecification": {
         "MaxValue": "10",
         "MinValue": "1"
       }
     },
     "Type": "Integer"
   }
   
   In the JSON, supply the following:
   
   i. For DefaultValue, specify a default value for the hyperparameter, if there is one.
   
   ii. For Description, specify a description for the hyperparameter.
   
   iii. For IsRequired, specify whether the hyperparameter is required.
   
   iv. For IsTunable, specify true if this hyperparameter can be tuned when a user runs a hyperparameter tuning job that uses this algorithm. For information, see Perform Automatic Model Tuning (p. 555).
   
   v. For Name, specify a name for the hyperparameter.
   
   vi. For Range, specify one of the following:

   - **IntegerParameterRangeSpecification** - the values of the hyperparameter are integers. Specify minimum and maximum values for the hyperparameter.
   
   - **ContinuousParameterRangeSpecification** - the values of the hyperparameter are floating-point values. Specify minimum and maximum values for the hyperparameter.
   
   - **CategoricalParameterRangeSpecification** - the values of the hyperparameter are categorical values. Specify a list of all of the possible values.
vii. For Type, specify Integer, Continuous, or Categorical. The value must correspond to the type of Range that you specified.

b. For Metric definitions, specify any training metrics that you want your algorithm to emit. Amazon SageMaker uses the regular expression that you specify to find the metrics by parsing the logs from your training container during training. Users can view these metrics when they run training jobs with your algorithm, and they can monitor and plot the metrics in Amazon CloudWatch. For information, see Monitor and Analyze Training Jobs Using Metrics (p. 587). For each metric, provide the following information:

i. For Metric name, type a name for the metric.

ii. For Regex, type the regular expression that Amazon SageMaker uses to parse training logs so that it can find the metric value.

iii. For Objective metric support choose Yes if this metric can be used as the objective metric for a hyperparameter tuning job. For information, see Perform Automatic Model Tuning (p. 555).

iv. Choose Add metric to add another metric, or choose Next if you are done adding metrics.

5. On the Inference specifications page, provide the following information if your algorithm supports inference:

a. For Container definition, type path in Amazon ECR where your inference container is stored.

b. For Container DNS host name, type the name of a DNS host for your image.

c. For Supported instance types for real-time inference, choose the instance types that your algorithm supports for models deployed as hosted endpoints in Amazon SageMaker. For information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 8).

d. For Supported instance types for batch transform jobs, choose the instance types that your algorithm supports for batch transform jobs. For information, see Get Inferences for an Entire Dataset with Batch Transform (p. 11).

e. For Supported content types, type the type of input data that your algorithm expects for inference requests.

f. For Supported response MIME types, type the MIME types that your algorithm supports for inference responses.

g. Choose Next.

6. On the Validation specifications page, provide the following information:

a. For Publish this algorithm on AWS Marketplace, choose Yes to publish the algorithm on AWS Marketplace.

b. For Validate this algorithm, choose Yes if you want Amazon SageMaker to run training jobs and/or batch transform jobs that you specify to test the training and/or inference code of your algorithm.

   **Note**
   To publish your algorithm on AWS Marketplace, your algorithm must be validated.

c. For IAM role, choose an IAM role that has the required permissions to run training jobs and batch transform jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 758).

d. For Validation profile, specify the following:

   - A name for the validation profile.
   - A Training job definition. This is a JSON block that describes a training job. This is in the same format as the TrainingJobDefinition (p. 1527) input parameter of the CreateAlgorithm (p. 854) API.
• A **Transform job definition.** This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition (p. 1538)` input parameter of the `CreateAlgorithm (p. 854)` API.

e. Choose **Create algorithm.**

---

Create an **Algorithm Resource (API)**

To create an algorithm resource by using the Amazon SageMaker API, call the `CreateAlgorithm (p. 854)` API.

Create a **Model Package Resource**

To create a model package resource that you can use to create deployable models in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker container that contains the inference code, or the algorithm resource that was used to train the model.
- The location of the model artifacts. Model artifacts can either be packaged in the same Docker container as the inference code or stored in Amazon S3.
- The instance types that your model package supports for both real-time inference and batch transform jobs.
- Validation profiles, which are batch transform jobs that Amazon SageMaker runs to test your model package's inference code.

Before listing model packages on AWS Marketplace, you must validate them. This ensures that buyers and sellers can be confident that products work in Amazon SageMaker. You can list products on AWS Marketplace only if validation succeeds.

The validation procedure uses your validation profile and sample data to run the following validations tasks:

1. Create a model in your account using the model package's inference image and the optional model artifacts that are stored in Amazon S3.

   **Note**
   
   A model package is specific to the region in which you create it. The S3 bucket where the model artifacts are stored must be in the same region where your created the model package.

2. Create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

3. Create a validation profile.

   **Note**
   
   In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. After fixing issues, recompute the algorithm. When the status of the algorithm is **COMPLETED**, find it in the Amazon SageMaker console and start the listing process.

   **Note**
   
   To publish your model package on AWS Marketplace, at least one validation profile is required.

You can create an model package either by using the Amazon SageMaker console or by using the Amazon SageMaker API.
Create a Model Package Resource (Console)

To create a model package in the Amazon SageMaker console:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Model packages, then choose Create model package.
3. On the Inference specifications page, provide the following information:
   a. For Model package name, type a name for your model package. The model package name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   b. Type a description for your model package. This description appears in the Amazon SageMaker console and in the AWS Marketplace.
   c. For Inference specification options, choose Provide the location of the inference image and model artifacts to create a model package by using an inference container and model artifacts. Choose Provide the algorithm used for training and its model artifacts to create a model package from an algorithm resource that you created or subscribe to from AWS Marketplace.
   d. If you chose Provide the location of the inference image and model artifacts for Inference specification options, provide the following information for Container definition and Supported resources:
      i. For Location of inference image, type the path to the image that contains your inference code. The image must be stored as a Docker container in Amazon ECR.
      ii. For Location of model data artifacts, type the location in S3 where your model artifacts are stored.
      iii. For Container DNS host name, type the name of the DNS host to use for your container.
      iv. For Supported instance types for real-time inference, choose the instance types that your model package supports for real-time inference from Amazon SageMaker hosted endpoints.
      v. For Supported instance types for batch transform jobs, choose the instance types that your model package supports for batch transform jobs.
      vi. Supported content types, type the content types that your model package expects for inference requests.
      vii. For Supported response MIME types, type the MIME types that your model package uses to provide inferences.
   e. If you chose Provide the algorithm used for training and its model artifacts for Inference specification options, provide the following information:
      i. For Algorithm ARN, type the Amazon Resource Name (ARN) of the algorithm resource to use to create the model package.
      ii. For Location of model data artifacts, type the location in S3 where your model artifacts are stored.
   f. Choose Next.
4. On the Validation and scanning page, provide the following information:
   a. For Publish this model package on AWS Marketplace, choose Yes to publish the model package on AWS Marketplace.
   b. For Validate this model package, choose Yes if you want Amazon SageMaker to run batch transform jobs that you specify to test the inference code of your model package.
Use Your Own Algorithms or Models

Note
To publish your model package on AWS Marketplace, your model package must be validated.

c. For IAM role, choose an IAM role that has the required permissions to run batch transform jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 758).

d. For Validation profile, specify the following:
   - A name for the validation profile.
   - A Transform job definition. This is a JSON block that describes a batch transform job. This is in the same format as the TransformJobDefinition (p. 1538) input parameter of the CreateAlgorithm (p. 854) API.

5. Choose Create model package.

Create a Model Package Resource (API)

To create a model package by using the Amazon SageMaker API, call the CreateModelPackage (p. 906) API.

Use Algorithm and Model Package Resources

You can create algorithms and model packages as resources in your Amazon SageMaker account, and you can find and subscribe to algorithms and model packages on AWS Marketplace.

Use algorithms to:
- Run training jobs. For information, see Use an Algorithm to Run a Training Job (p. 492).
- Run hyperparameter tuning jobs. For information, see Use an Algorithm to Run a Hyperparameter Tuning Job (p. 494).
- Create model packages. After you use an algorithm resource to run a training job or a hyperparameter tuning job, you can use the model artifacts that these jobs output along with the algorithm to create a model package. For information, see Create a Model Package Resource (p. 489).

Note
If you subscribe to an algorithm on AWS Marketplace, you must create a model package before you can use it to get inferences by creating hosted endpoint or running a batch transform job.
Use model packages to:

- Create models that you can use to get real-time inference or run batch transform jobs. For information, see Use a Model Package to Create a Model (p. 497).
- Create hosted endpoints to get real-time inference. For information, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35).
- Create batch transform jobs. For information, see Step 6.2: Deploy the Model with Batch Transform (p. 37).

Topics
- Use an Algorithm to Run a Training Job (p. 492)
- Use an Algorithm to Run a Hyperparameter Tuning Job (p. 494)
- Use a Model Package to Create a Model (p. 497)

Use an Algorithm to Run a Training Job

You can create use an algorithm resource to create a training job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics
- Use an Algorithm to Run a Training Job (Console) (p. 493)
- Use an Algorithm to Run a Training Job (API) (p. 494)
Use an Algorithm to Run a Training Job (Console)

To use an algorithm to run a training job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms.
3. Choose an algorithm that you created from the list on the My algorithms tab or choose an algorithm that you subscribed to on the AWS Marketplace subscriptions tab.
4. Choose Create training job.

The algorithm you chose will automatically be selected.

5. On the Create training job page, provide the following information:

   a. For Job name, type a name for the training job.

   b. For IAM role, choose an IAM role that has the required permissions to run training jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 758).

   c. For Resource configuration, provide the following information:

      i. For Instance type, choose the instance type to use for training.

      ii. For Instance count, type the number of ML instances to use for the training job.

      iii. For Additional volume per instance (GB), type the size of the ML storage volume that you want to provision. ML storage volumes store model artifacts and incremental states.

      iv. For Encryption key, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instance, specify the key.

      v. For Stopping condition, specify the maximum amount of time in seconds, minutes, hours, or days, that you want the training job to run.

   d. For VPC, choose a Amazon VPC that you want to allow your training container to access. For more information, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 789).

   e. For Hyperparameters, specify the values of the hyperparameters to use for the training job.

   f. For Input data configuration, specify the following values for each channel of input data to use for the training job. You can see what channels the algorithm you're using for training sports, and the content type, supported compression type, and supported input modes for each channel, under Channel specification section of the Algorithm summary page for the algorithm.

      i. For Channel name, type the name of the input channel.

      ii. For Content type, type the content type of the data that the algorithm expects for the channel.

      iii. For Compression type, choose the data compression type to use, if any.

      iv. For Record wrapper, choose RecordIO if the algorithm expects data in the RecordIO format.

      v. For S3 data type, S3 data distribution type, and S3 location, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 1501).

      vi. For Input mode, choose File to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose PipeTo stream data directly from Amazon S3 to the container.
vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.

g. For **Output** location, specify the following values:

i. For **S3 output path**, choose the S3 location where the training job stores output, such as model artifacts.
   **Note**
   You use the model artifacts stored at this location to create a model or model package from this training job.

ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

h. For **Tags**, specify one or more tags to manage the training job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

i. Choose **Create training job** to run the training job.

**Use an Algorithm to Run a Training Job (API)**

To use an algorithm to run a training job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) as the **AlgorithmName** field of the **AlgorithmSpecification** object that you pass to **CreateTrainingJob**. For information about training models in Amazon SageMaker, see **Train a Model with Amazon SageMaker**.

**Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK)**

Use an algorithm that you created or subscribed to on AWS Marketplace to create a training job, create an **AlgorithmEstimator** object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the **algorithm_arn** argument. Then call the **fit** method of the estimator. For example:

```python
from sagemaker import AlgorithmEstimator

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(  
    role='SageMakerRole',  
    train_instance_count=1,  
    train_instance_type='ml.c4.xlarge',  
    sagemaker_session=sagemaker_session,  
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(  
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

**Use an Algorithm to Run a Hyperparameter Tuning Job**

A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. For more information, see **Perform Automatic Model Tuning**.

You can create use an algorithm resource to create a hyperparameter tuning job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

**Topics**
Use an Algorithm to Run a Hyperparameter Tuning Job (Console) (p. 495)
• Use an Algorithm to Run a Hyperparameter Tuning Job (API) (p. 496)
• Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK) (p. 497)

Use an Algorithm to Run a Hyperparameter Tuning Job (Console)
To use an algorithm to run a hyperparameter tuning job (console)
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms.
3. Choose an algorithm that you created from the list on the My algorithms tab or choose an algorithm that you subscribed to on the AWS Marketplace subscriptions tab.
4. Choose Create hyperparameter tuning job.

The algorithm you chose will automatically be selected.
5. On the Create hyperparameter tuning job page, provide the following information:

a. For Warm start, choose Enable warm start to use the information from previous hyperparameter tuning jobs as a starting point for this hyperparameter tuning job. For more information, see Run a Warm Start Hyperparameter Tuning Job (p. 570).

i. Choose Identical data and algorithm if your input data is the same as the input data for the parent jobs of this hyperparameter tuning job, or choose Transfer learning to use additional or different input data for this hyperparameter tuning job.

ii. For Parent hyperparameter tuning job(s), choose up to 5 hyperparameter tuning jobs to use as parents to this hyperparameter tuning job.

b. For Hyperparameter tuning job name, type a name for the tuning job.

c. For IAM role, choose an IAM role that has the required permissions to run hyperparameter tuning jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 758).

d. For VPC, choose a Amazon VPC that you want to allow the training jobs that the tuning job launches to access. For more information, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 789).

e. Choose Next.

f. For Objective metric, choose the metric that the hyperparameter tuning job uses to determine the best combination of hyperparameters, and choose whether to minimize or maximize this metric. For more information, see View the Best Training Job (p. 567).

g. For Hyperparameter configuration, choose ranges for the tunable hyperparameters that you want the tuning job to search, and set static values for hyperparameters that you want to remain constant in all training jobs that the hyperparameter tuning job launches. For more information, see Define Hyperparameter Ranges (p. 558).

h. Choose Next.

i. For Input data configuration, specify the following values for each channel of input data to use for the hyperparameter tuning job. You can see what channels the algorithm you're using for hyperparameter tuning supports, and the content type, supported compression type, and supported input modes for each channel, under Channel specification section of the Algorithm summary page for the algorithm.

i. For Channel name, type the name of the input channel.

ii. For Content type, type the content type of the data that the algorithm expects for the channel.

iii. For Compression type, choose the data compression type to use, if any.
iv. For Record wrapper, choose RecordIO if the algorithm expects data in the RecordIO format.

v. For S3 data type, S3 data distribution type, and S3 location, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 1501).

vi. For Input mode, choose File to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose Pipe to stream data directly from Amazon S3 to the container.

vii. To add another input channel, choose Add channel. If you are finished adding input channels, choose Done.

j. For Output location, specify the following values:
   i. For S3 output path, choose the S3 location where the training jobs that this hyperparameter tuning job launches store output, such as model artifacts.

   Note
   You use the model artifacts stored at this location to create a model or model package from this hyperparameter tuning job.

   ii. For Encryption key, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

k. For Resource configuration, provide the following information:
   i. For Instance type, choose the instance type to use for each training job that the hyperparameter tuning job launches.

   ii. For Instance count, type the number of ML instances to use for each training job that the hyperparameter tuning job launches.

   iii. For Additional volume per instance (GB), type the size of the ML storage volume that you want to provision each training job that the hyperparameter tuning job launches. ML storage volumes store model artifacts and incremental states.

   iv. For Encryption key, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instances, specify the key.

l. For Resource limits, provide the following information:
   i. For Maximum training jobs, specify the maximum number of training jobs that you want the hyperparameter tuning job to launch. A hyperparameter tuning job can launch a maximum of 500 training jobs.

   ii. For Maximum parallel training jobs, specify the maximum number of concurrent training jobs that the hyperparameter tuning job can launch. A hyperparameter tuning job can launch a maximum of 10 concurrent training jobs.

   iii. For Stopping condition, specify the maximum amount of time in seconds, minutes, hours, or days, that you want each training job that the hyperparameter tuning job launches to run.

   m. For Tags, specify one or more tags to manage the hyperparameter tuning job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

   n. Choose Create jobs to run the hyperparameter tuning job.

Use an Algorithm to Run a Hyperparameter Tuning Job (API)

To use an algorithm to run a hyperparameter tuning job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) of the algorithm as the AlgorithmName field of the AlgorithmSpecification (p. 1274) object that you pass to CreateHyperParameterTuningJob (p. 890). For information about hyperparameter tuning in Amazon SageMaker, see Perform Automatic Model Tuning (p. 555).
Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a hyperparameter tuning job, create an AlgorithmEstimator object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the algorithm_arn argument. Then initialize a HyperparameterTuner object with the AlgorithmEstimator you created as the value of the estimator argument. Finally, call the fit method of the AlgorithmEstimator. For example:

```python
from sagemaker import AlgorithmEstimator
from sagemaker.tuner import HyperparameterTuner

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.set_hyperparameters(max_leaf_nodes=10)

tuner = HyperparameterTuner(estimator=algo, base_tuning_job_name='some-name',
    objective_metric_name='validation:accuracy',
    hyperparameter_ranges=hyperparameter_ranges,
    max_jobs=2, max_parallel_jobs=2)

# Launch the Hyperparameter Tuning Job
# tuner.fit({'training': train_input}, include_cls_metadata=False)

tuner.wait()
```

Use a Model Package to Create a Model

Use a model package to create a deployable model that you can use to get real-time inferences by creating a hosted endpoint or to run batch transform jobs. You can create a deployable model from a model package by using the Amazon SageMaker console, the low-level Amazon SageMaker API), or the Amazon SageMaker Python SDK.

Topics

- Use a Model Package to Create a Model (Console) (p. 497)
- Use a Model Package to Create a Model (API) (p. 498)
- Use a Model Package to Create a Model (Amazon SageMaker Python SDK) (p. 498)

Use a Model Package to Create a Model (Console)

To create a deployable model from a model package (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Model packages**.
3. Choose a model package that you created from the list or choose a model package that you subscribed to from the **AWS Marketplace subscriptions** tab.
4. Choose **Create model**.
5. For **Model name**, type a name for the model.
6. For IAM role, choose an IAM role that has the required permissions to call other services on your behalf, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 758).

7. For VPC, choose a Amazon VPC that you want to allow the model to access. For more information, see Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC (p. 792).

8. Leave the default values for Container input options and Choose model package.

9. For environment variables, provide the names and values of environment variables you want to pass to the model container.

10. For Tags, specify one or more tags to manage the model. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see AWS Tagging Strategies.

11. Choose Create model.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 37).

Use a Model Package to Create a Model (API)

To use a model package to create a deployable model by using the Amazon SageMaker API, specify the name or the Amazon Resource Name (ARN) of the model package as the ModelPackageName field of the ContainerDefinition (p. 1321) object that you pass to the CreateModel (p. 902) API.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 37).

Use a Model Package to Create a Model (Amazon SageMaker Python SDK)

To use a model package to create a deployable model by using the Amazon SageMaker Python SDK, initialize a ModelPackage object, and pass the Amazon Resource Name (ARN) of the model package as the model_package_arn argument. For example:

```python
from sagemaker import ModelPackage
model = ModelPackage(role='SageMakerRole',
                     model_package_arn='training-job-scikit-decision-trees-1542660466-6f92',
                     sagemaker_session=sagemaker_session)
```

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35). For information about batch transform jobs, see Step 6.2: Deploy the Model with Batch Transform (p. 37).
Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace

Amazon SageMaker integrates with AWS Marketplace, enabling developers to charge other Amazon SageMaker users for the use of their algorithms and model packages. AWS Marketplace is a curated digital catalog that makes it easy for customers to find, buy, deploy, and manage third-party software and services that customers need to build solutions and run their businesses. AWS Marketplace includes thousands of software listings in popular categories, such as security, networking, storage, machine learning, business intelligence, database, and DevOps. It simplifies software licensing and procurement with flexible pricing options and multiple deployment methods. For information, see AWS Marketplace Documentation.

Topics

- Amazon SageMaker Algorithms (p. 499)
- Amazon SageMaker Model Packages (p. 499)
- Sell Amazon SageMaker Algorithms and Model Packages (p. 499)
- Find and Subscribe to Algorithms and Model Packages on AWS Marketplace (p. 502)
- Use Algorithm and Model Package Resources (p. 491)

Amazon SageMaker Algorithms

An algorithm enables you to perform end-to-end machine learning. It has two logical components: training and inference. Buyers can use the training component to create training jobs in Amazon SageMaker and build a machine learning model. Amazon SageMaker saves the model artifacts generated by the algorithm during training to an Amazon S3 bucket. For more information, see Train a Model with Amazon SageMaker (p. 5).

Buyers use the inference component with the model artifacts generated during a training job to create a deployable model in their Amazon SageMaker account. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 8).

Amazon SageMaker Model Packages

Buyers use a model package to build a deployable model in Amazon SageMaker. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 8). As a seller, you can build your model artifacts by training in Amazon SageMaker, or you can use your own model artifacts from a model that you trained outside of Amazon SageMaker. You can charge buyers for inference.

Sell Amazon SageMaker Algorithms and Model Packages

Selling Amazon SageMaker algorithms and model packages is a three-step process:

1. Develop your algorithm or model, and package it in a Docker container. For information, see Develop Algorithms and Models in Amazon SageMaker (p. 500).
2. Create an algorithm or model package resource in Amazon SageMaker. For information, see Create Algorithm and Model Package Resources (p. 485).

3. Register as a seller on AWS Marketplace and list your algorithm or model package on AWS Marketplace. For information about registering as a seller, see Getting Started as a Seller in the User Guide for AWS Marketplace Providers. For information about listing and monetizing your algorithms and model packages, see Listing Algorithms and Model Packages in AWS Marketplace for Machine Learning in the User Guide for AWS Marketplace Providers.

Topics

- Develop Algorithms and Models in Amazon SageMaker (p. 500)
- Create Algorithm and Model Package Resources (p. 485)
- List Your Algorithm or Model Package on AWS Marketplace (p. 501)

Develop Algorithms and Models in Amazon SageMaker

Before you can create algorithm and model package resources to use in Amazon SageMaker or list on AWS Marketplace, you have to develop them and package them in Docker containers.

**Note**

When algorithms and model packages are created for listing on AWS Marketplace, Amazon SageMaker scans the containers for security vulnerabilities on supported operating systems. Only the following operating system versions are supported:

- Debian: 6.0, 7, 8, 9, 10
- CentOS: 5, 6, 7
- Oracle Linux: 5, 6, 7
- Alpine: 3.3, 3.4, 3.5
- Amazon Linux
Develop Algorithms in Amazon SageMaker

An algorithm should be packaged as a docker container and stored in Amazon ECR to use it in Amazon SageMaker. The Docker container contains the training code used to run training jobs and, optionally, the inference code used to get inferences from models trained by using the algorithm.

For information about developing algorithms in Amazon SageMaker and packaging them as containers, see Use Your Own Algorithms or Models with Amazon SageMaker (p. 456). For a complete example of how to create an algorithm container, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the Advanced Functionality section, and is named scikit_bring_your_own.ipynb. For information about using the sample notebooks in a notebook instance, see Use Example Notebooks (p. 208).

Always thoroughly test your algorithms before you create algorithm resources to publish on AWS Marketplace.

Note
When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

Develop Models in Amazon SageMaker

A deployable model in Amazon SageMaker consists of inference code, model artifacts, an IAM role that is used to access resources, and other information required to deploy the model in Amazon SageMaker. Model artifacts are the results of training a model by using a machine learning algorithm. The inference code must be packaged in a Docker container and stored in Amazon ECR. You can either package the model artifacts in the same container as the inference code, or store them in Amazon S3.

You create a model by running a training job in Amazon SageMaker, or by training a machine learning algorithm outside of Amazon SageMaker. If you run a training job in Amazon SageMaker, the resulting model artifacts are available in the ModelArtifacts field in the response to a call to the DescribeTrainingJob (p. 1066) operation. For information about how to develop an Amazon SageMaker model container, see Use Your Own Inference Code (p. 480). For a complete example of how to create a model container from a model trained outside of Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/xgboost_bring_your_own_model/xgboost_bring_your_own_model.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the Advanced Functionality section, and is named xgboost_bring_your_own_model.ipynb. For information about using the sample notebooks in a notebook instance, see Use Example Notebooks (p. 208).

Always thoroughly test your models before you create model packages to publish on AWS Marketplace.

Note
When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

List Your Algorithm or Model Package on AWS Marketplace

After creating and validating your algorithm or model in Amazon SageMaker, list your product on AWS Marketplace. The listing process makes your products available in the AWS Marketplace and the Amazon SageMaker console.
To list products on AWS Marketplace, you must be a registered seller. To register, use the self-registration process from the AWS Marketplace Management Portal (AMMP). For information, see Getting Started as a Seller in the User Guide for AWS Marketplace Providers. When you start the product listing process from the Amazon SageMaker console, we check your seller registration status. If you have not registered, we direct you to do so.

To start the listing process, do one of the following:

- From the Amazon SageMaker console, choose the product, choose Actions, and choose Publish new ML Marketplace listing. This carries over your product reference, the Amazon Resource Name (ARN), and directs you to the AMMP to create the listing.
- Go to ML listing process, manually enter the Amazon Resource Name (ARN), and start your product listing. This process carries over the product metadata that you entered when creating the product in Amazon SageMaker. For an algorithm listing, the information includes the supported instance types and hyperparameters. In addition, you can enter a product description, promotional information, and support information as you would with other AWS Marketplace products.

Find and Subscribe to Algorithms and Model Packages on AWS Marketplace

With AWS Marketplace, you can browse and search for hundreds of machine learning algorithms and models in a broad range of categories, such as computer vision, natural language processing, speech recognition, text, data, voice, image, video analysis, fraud detection, predictive analysis, and more.

To find algorithms on AWS Marketplace
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms, then choose Find algorithms.

This takes you to the AWS Marketplace algorithms page. For information about finding and subscribing to algorithms on AWS Marketplace, see Machine Learning Products in the AWS Marketplace User Guide for AWS Consumers.

To find model packages on AWS Marketplace
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Model packages, then choose Find model packages.

This takes you to the AWS Marketplace model packages page. For information about finding and subscribing to model packages on AWS Marketplace, see Machine Learning Products in the AWS Marketplace User Guide for AWS Consumers.
Use Algorithms and Model Packages

For information about using algorithms and model packages that you subscribe to in Amazon SageMaker, see Use Algorithm and Model Package Resources (p. 491).

Note
When you create a training job, inference endpoint, and batch transform job from an algorithm or model package that you subscribe to on AWS Marketplace, the training and inference containers do not have access to the internet. Because the containers do not have access to the internet, the seller of the algorithm or model package does not have access to your data.
Train Models

For an overview on training models with Amazon SageMaker, see Train a Model with Amazon SageMaker (p. 5).

Amazon SageMaker provides features to monitor and manage the training and validation of machine learning models. For guidance on metrics available, incremental training, automatic model tuning, and the use of augmented manifest files to label training data, see the following topics.

- For guidance on debugging the training of machine learning models, see Amazon SageMaker Debugger (p. 516).
- For guidance on metrics used to monitor and train models, see Monitor and Analyze Training Jobs Using Metrics (p. 587).
- For guidance on incremental training in Amazon SageMaker, see Incremental Training in Amazon SageMaker (p. 594).
- For guidance on using managed spot training in Amazon SageMaker, see Managed Spot Training in Amazon SageMaker (p. 599).
- For guidance on using training checkpoints in Amazon SageMaker, see Use Checkpoints in Amazon SageMaker (p. 600).
- For guidance on automatic model tuning, also known as hyperparameter tuning, see Perform Automatic Model Tuning (p. 555).
- For guidance on using an augmented manifest file to label training data, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600).

Topics
- Manage Machine Learning with Amazon SageMaker Experiments (p. 504)
- Amazon SageMaker Debugger (p. 516)
- Perform Automatic Model Tuning (p. 555)
- Tune Multiple Algorithms to Find the Best Model (p. 575)
- Use Reinforcement Learning with Amazon SageMaker (p. 578)
- Train a Deep Graph Network (p. 585)
- Monitor and Analyze Training Jobs Using Metrics (p. 587)
- Incremental Training in Amazon SageMaker (p. 594)
- Managed Spot Training in Amazon SageMaker (p. 599)
- Use Checkpoints in Amazon SageMaker (p. 600)
- Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 600)

Manage Machine Learning with Amazon SageMaker Experiments

Amazon SageMaker Experiments is a capability of Amazon SageMaker that lets you organize, track, compare, and evaluate your machine learning experiments.

Machine learning is an iterative process. You need to experiment with multiple combinations of data, algorithm and parameters, all the while observing the impact of incremental changes on model accuracy. Over time this iterative experimentation can result in thousands of model training runs and model versions. This makes it hard to track the best performing models and their input configurations. It's
also difficult to compare active experiments with past experiments to identify opportunities for further incremental improvements.

Amazon SageMaker Experiments automatically tracks the inputs, parameters, configurations, and results of your iterations as *trials*. You can assign, group, and organize these trials into *experiments*. Experiments is integrated with Amazon SageMaker Studio providing a visual interface to browse your active and past experiments, compare trials on key performance metrics, and identify the best performing models.

Amazon SageMaker Experiments comes with the Amazon SageMaker Python SDK which makes the search and analytics capabilities easily accessible in Amazon SageMaker Notebooks. Because Experiments enables tracking of all the steps and artifacts that went into creating a model, you can quickly revisit the origins of a model when you are troubleshooting issues in production, or auditing your models for compliance verifications.

**Topics**
- Organize Experiments (p. 505)
- Track Experiments (p. 505)
- Compare and Evaluate Experiments (p. 506)
- Amazon SageMaker Autopilot (p. 506)
- Track and Evaluate a Model Training Experiment (p. 506)
- Search (p. 511)

**Organize Experiments**

Amazon SageMaker Experiments offers a structured organization scheme to help users group and organize their machine learning iterations. The top level entity, an Amazon SageMaker *experiment*, is a collection of *trials* that are observed, compared, and evaluated as a group. A trial is a set of steps called *trial components*. Each trial component can include a combination of inputs such as datasets, algorithms, and parameters, and produce specific outputs such as models, metrics, datasets, and checkpoints. Examples of trial components are data pre-processing jobs, training jobs, and batch transform jobs.

The goal of an experiment is to determine the trial that produces the best model. Multiple trials are performed, each one isolating and measuring the impact of a change to one or more inputs, while keeping the remaining inputs constant. By analyzing the trials, you can determine which features have the most effect on the model.

**Track Experiments**

Amazon SageMaker Experiments enables tracking of experiments.

**Automated Tracking**

Amazon SageMaker Experiments automatically tracks Amazon SageMaker Autopilot jobs as experiments with their underlying training jobs tracked as trials. Experiments also automatically tracks Amazon SageMaker independently executed training, batch transform, and processing jobs as trial components, whether assigned to a trial or left unassigned. Unassigned trial components can be associated with a trial at a later time. All experiment artifacts including datasets, algorithms, hyperparameters, and model metrics are tracked and recorded. This data allows customers to trace the complete lineage of a model which helps with model governance, auditing, and compliance verifications.

**Manual Tracking**

Amazon SageMaker Experiments provides tracking APIs in the Amazon SageMaker Python SDK for recording and tracking machine learning workflows running locally on Amazon SageMaker Studio.
Compare and Evaluate Experiments

Amazon SageMaker Experiments is integrated with Amazon SageMaker Studio. When you use Studio, Experiments automatically tracks your experiments and trials, and presents visualizations of the tracked data and an interface to search the data.

Amazon SageMaker Experiments automatically organizes, ranks, and sorts trials based on a chosen metric using the concept of a trial leaderboard. Amazon SageMaker Studio produces real-time data visualizations, such as metric charts and graphs, to quickly compare and identify the best performing models. These are updated in real-time as the experiment progresses.

Amazon SageMaker Autopilot

Amazon SageMaker Experiments is integrated with Amazon SageMaker Autopilot. When you perform an automated machine learning job using Autopilot, Experiments creates an experiment for the job, and trials for each of the different combinations of trial components, parameters, and artifacts that Autopilot tries for the job. You can visually drill into all trials and components using Amazon SageMaker Studio.

Track and Evaluate a Model Training Experiment

This tutorial demonstrates how to visually track, compare, and evaluate a model training experiment using Amazon SageMaker Studio. The basis of the tutorial is the MNIST Handwritten Digits Classification Experiment (MNIST) example notebook.

It is intended that this topic be viewed alongside Studio with the MNIST notebook open. As you run through the cells, the sections highlight the relevant code and show you how to observe the results in Studio. Some of the code snippets have been edited for brevity.

Prerequisites

- A local copy of the MNIST example notebook and the companion mnist.py file. Both are available from the awslabs/amazon-sagemaker-examples repository. To download the files, select each link, right-click on the Raw button, and then choose Save as.
- An AWS SSO or IAM account to sign-on to Amazon SageMaker Studio. For more information, see Onboard to Amazon SageMaker Studio (p. 16).

Topics

- Open the Notebook in Studio (p. 506)
- Set Up Amazon SageMaker Experiments (p. 507)
- Create and Track an Experiment (p. 507)
- Compare Trials and Analyze (p. 509)
- Deploy the Top Model (p. 510)
- Clean Up Resources (p. 510)

Open the Notebook in Studio

To open the notebook

1. Sign-on to Studio.
2. In the left sidebar, select the File Browser icon.
3. At the top of the File Browser, select the **Up arrow** icon and then a **File Upload** dialog opens. Browse to and select your local versions of the *mnist-handwritten-digits-classification-experiment.ipynb* and *mnist.py* files, and then choose **Open**.

4. Double-click the uploaded notebook file to open the notebook in a new tab.

### Set Up Amazon SageMaker Experiments

The Amazon SageMaker Experiments SDK is separate from the Amazon SageMaker Python SDK. In the following steps you install the Experiments SDK and import the relevant modules. For more information on the Experiments SDK, see [sagemaker-experiments](https://aws.amazon.com/sagemaker-experiments).

**To install the SDK and import modules**

1. Install the Experiments SDK.

   ```bash
   !{sys.executable} -m pip install sagemaker-experiments
   ```

2. Import the Amazon SageMaker and Experiments modules.

   ```python
   import sagemaker
   from sagemaker import get_execution_role
   from sagemaker.session import Session
   from sagemaker.analytics import ExperimentAnalytics
   from smexperiments.experiment import Experiment
   from smexperiments.trial import Trial
   from smexperiments.trial_component import TrialComponent
   from smexperiments.tracker import Tracker
   ```

3. Run the remaining cells until you come to the last cell in the **Dataset** section.

### Create and Track an Experiment

The following code creates and tracks an experiment. First, a Tracker is created and used to track the dataset transform job. Next, an experiment is created and then 5 trials are created inside a loop, one for each value of the `num_hidden_channel` hyperparameter you're testing.

1. In the left sidebar, select the **SageMaker Experiment List** icon to display the experiments browser.

2. In the notebook, create a Tracker as a preprocessing step to track the transform job for the dataset. The tracker logs the normalization parameters and the URI to the Amazon S3 bucket where the transformed dataset is uploaded.

   ```python
   with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
       tracker.log_parameters({
           "normalization_mean": 0.1307,
           "normalization_std": 0.3081,
       })
       tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
   
   preprocessing_trial_component = tracker.trial_component
   ```

After the previous code runs, the experiments list contains an entry named **Unassigned trial components**. This is the preprocessing step just created. Your screen should look similar to the following:
3. Create and start tracking an experiment.

```python
mnist_experiment = Experiment.create(
    experiment_name=f"mnist-hand-written-digits-classification-{int(time.time())}",
    description="Classification of mnist hand-written digits",
    sagemaker_boto_client=sm)
print(mnist_experiment)
```

Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f7152b326d8>,
    experiment_name='mnist-hand-written-digits-classification-1575947870',
    description='Classification of mnist hand-written digits',

After the previous code runs, the experiments list contains an entry for the experiment. It might take a moment to display. Your screen should look similar to the following:

4. Double-click on your experiment to display a list of the trials in the experiment. Initially the list is empty.

5. Create trials for the experiment. Each trial trains a model using a different number for the `hidden_channels` hyperparameter. The preprocessing trial component is added to each trial for complete tracking (for example, for auditing purposes). The code also specifies definitions for the following metrics:

- Train loss
- Test loss
- Test accuracy

The definitions tell Amazon SageMaker to capture those metrics from the algorithm's log output. The metrics are used later to evaluate and compare the models.
preprocessing_trial_component = tracker.trial_component

for i, num_hidden_channel in enumerate([2, 5, 10, 20, 32]):
    trial_name = f"cnn-training-job-{num_hidden_channel}-hidden-channels-{int(time.time())}"  
cnn_trial = Trial.create(
    trial_name=trial_name,
    experiment_name=mnist_experiment.experiment_name,
    sagemaker_boto_client=sm,
)
hidden_channel_trial_name_map[num_hidden_channel] = trial_name

cnn_trial.add_trial_component(preprocessing_trial_component)

estimator = PyTorch(
    hyperparameters={
        'hidden_channels': num_hidden_channel,
    },
    metric_definitions=[[  
        {  
            'Name': 'train:loss', 'Regex': 'Train Loss: (.*)'},  
        {  
            'Name': 'test:loss', 'Regex': 'Test Average loss: (.*)'},  
        {  
            'Name': 'test:accuracy', 'Regex': 'Test Accuracy: (.*%);'}
        ],
    enable_sagemaker_metrics=True,
)

cnn_training_job_name = "cnn-training-job-{}.format(int(time.time()))"

estimator.fit(
    inputs={'training': inputs},
    job_name=cnn_training_job_name,
    experiment_config={
        "TrialName": cnn_trial.trial_name,
        "TrialComponentDisplayName": "Training",
    },
)

The trial list automatically updates as each training job runs. It takes a few minutes for each trial to be displayed. Your screen should look similar to the following:

![Screen capture showing the trial list in Amazon SageMaker Studio UI](image)

### Compare Trials and Analyze

This section deviates from the notebook and show you how to compare and analyze the trained models using the Amazon SageMaker Studio UI.
To view a list of training jobs ordered by test:accuracy

1. Select all 5 trials, right-click the selection, and then choose Open in trial component list. A new tab opens that displays a list of the components for all trials. There's a preprocessing job and training job for each trial.
2. If the TABLE PROPERTIES pane isn't open, select the gear icon in the upper right corner to open it. Unselect the Created on and Last modified checkboxes.
3. Right-click the test:accuracy column header, choose Data aggregation, and then choose Maximum. Choose the test:accuracy column header to sort the list by decreasing maximum test accuracy. You can see that the models trained with the hidden_channels hyperparameter set to 2 and 20 give the highest test accuracy. Due to the randomness of model training and the closeness of the accuracies, your results might differ. Your screen should look similar to the following:

![Image of trial components list]

Deploy the Top Model

Now that you've determined the most accurate training job, deploy the associated model to an Amazon SageMaker endpoint.

To deploy the model

1. In the left sidebar, select the SageMaker Endpoint List icon to display the endpoints browser.
2. Right-click the top training job in the trial components list and choose Deploy model. A new tab opens that displays the Deploy model page.
3. Under REQUIRED SETTINGS, enter a name for the endpoint. Keep the default values of ml.m5.xlarge for Instance type and 1 for Instance count.
4. Choose Deploy model.

Clean Up Resources

To avoid incurring unnecessary charges, delete the resources you created after you're done with the tutorial. You can't delete the Experiment resources through the Amazon SageMaker Management Console or the Studio UI. The following code shows how to clean up these resources.

To delete the experiment, first you must delete all trials in the experiment. To delete a trial, first you must remove all trial components from the trial. To delete a trial component, first you must remove the component from all trials.
Note
Trial components can exist independent of trials and experiments. You might want keep them if you plan on further exploration. If so, comment out `tc.delete()`

```python
def cleanup(experiment):
    for trial_summary in experiment.list_trials():
        trial = Trial.load(sagemaker_boto_client=sm, trial_name=trial_summary.trial_name)
        for trial_component_summary in trial.list_trial_components():
            tc = TrialComponent.load(
                sagemaker_boto_client=sm,
                trial_component_name=trial_component_summary.trial_component_name)
            trial.remove_trial_component(tc)
            try:
                # comment out to keep trial components
                tc.delete()
            except:
                # tc is associated with another trial
                continue
                # to prevent throttling
            time.sleep(.5)
            trial.delete()
    cleanup(mnist_experiment)
```

For information on deleting your Amazon S3 buckets, see How Do I Delete an S3 Bucket?

To delete the notebook
1. Select the file browser.
2. Right-click the notebook file and choose Shut Down Kernel.
3. Right-click the notebook file and choose Delete.

Search
Developing a machine learning model typically requires extensive experimenting with different datasets, algorithms, and hyperparameter values. To manage up to thousands of machine learning model experiments, use Amazon SageMaker’s search capabilities.

You can use Amazon SageMaker search to:

- Organize, find, and evaluate training jobs using properties, hyperparameters, performance metrics, or any metadata.
- Find the best performing model by reviewing training job and model metrics, such as training loss or validation accuracy.
- Trace a model’s lineage to the training job and its related resources, such as the training datasets.

You can organize, find, and evaluate training jobs and models using the Amazon SageMaker console or the API.

Topics
- Sample Notebooks for Managing ML Experiments (p. 512)
- Organize, Find, and Evaluate Training Jobs (Console) (p. 512)
- Find and Evaluate Training Jobs (API) (p. 514)
- Verify the Datasets Used by Your Training Jobs (p. 515)
Sample Notebooks for Managing ML Experiments

For a sample notebook that uses Amazon SageMaker model tracking capability to manage ML experiments, see Managing ML Experimentation using Amazon SageMaker Model Tracking Capability.

For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After you have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all of the Amazon SageMaker samples. The notebook for managing ML experiments is located in the Advanced Functionality section. To open a notebook, choose its Use tab, and choose Create copy. If you have questions, post them on the Amazon Machine Learning Developer Forum.

Organize, Find, and Evaluate Training Jobs (Console)

To organize training jobs, assign one or more tags to them.

To find a specific training job, model, or resource, use model tracking to search on keywords assigned to any searchable items. Searchable items include training jobs, models, hyperparameters, metadata, tags, and URLs. To refine your tracking results, you can search using multiple criteria.

To choose the best model for deployment, evaluate how all models performed against one or more metrics. You can use model tracking results to list, sort, and evaluate the performance of the models in your experiments.

Use Tags to Track Training Jobs (Console)

To group training jobs, create tags with descriptive keys and a value. For example, create tag keys for: project, owner, customer, and industry.

Add tags to training jobs (console)

2. In the navigation pane, choose Training jobs and Create training job.
3. Scroll to the bottom of the page and enter a key and value for the tag.
4. To add another tag, choose Add tag, and add another key-value pair.
Find Training Jobs (Console)

You can search for training jobs using a variety of job attributes. Note that some search parameters appear only if you have created a training job with that attribute. For example, Tags appears only if you have added a tag for a training job.

To find training jobs (console)

2. In the navigation pane, choose Search.
3. Add Parameters.
   a. In the search box, enter a parameter and choose a parameter type, for example TrainingJobName.
   b. Choose a conditional operation. For numeric values, use operators such as is equals to, lesser than, or or greater than. For text-based values, use operators such as equals to or contains.
   c. Enter a value for the parameter.
4. (Optional) To refine your search, add additional search criteria. Choose Add row and enter the parameter values.
5. Choose Search.

Evaluate Models (Console)

To evaluate a model's performance, review its metadata, hyperparameters, and metrics. To highlight metrics, adjust the view to show only metrics and important hyperparameters.

To evaluate a model (console)

2. In the navigation pane, choose Search and search for training jobs by specifying relevant parameters. The results are displayed in a table.
3. Open the preferences window by choosing the settings icon in the search results table.
4. To show or hide a hyperparameter or metric, turn it on or off by choosing **Hyperparameter** or **Metric**.

5. Make necessary changes, then choose **Update view**.

6. After viewing metrics and important hyperparameters, you can compare and contrast the result. Then, you can choose the best model to host or investigate the models that are performing poorly.

### Find and Evaluate Training Jobs (API)

To find and evaluate training jobs or to get suggestions for items used in experiments that are searchable, you can use the [Search](p. 1196) API.

**Topics**

- Find Training Jobs (API) (p. 514)
- Evaluate Models (API) (p. 514)
- Get Suggestions for a Search (API) (p. 515)

#### Find Training Jobs (API)

To find training jobs, create a search parameter using the `search_params` parameter. Then use the search function in the `smclient` subprocess in the AWS SDK for Python (Boto 3).

The following example shows how to use the [Search](p. 1196) API to find training jobs.

```python
import boto3

search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [{
            "Name": "Tags.Project",
            "Operator": "Equals",
            "Value": "Project_Binary_Classifier"
        }],
        "SortBy": "Metrics.train:binary_classification_accuracy",
        "SortOrder": "Descending"
    }
}

smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)
```

#### Evaluate Models (API)

To evaluate models, run a search as described in Find Training Jobs (API) (p. 514), review model metrics, then, use the AWS SDK for Python (Boto 3) to create a table and plot it.

The following example shows how to evaluate models and to display the results in a table.

```python
import pandas

headers=["Training Job Name", "Training Job Status", "Batch Size", "Binary Classification Accuracy"]
rows=[]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
    rows.append([trainingJob['TrainingJobName'],
                  trainingJob['Status'],
                  trainingJob['BatchSize'],
                  metrics['binary_classification_accuracy']])

df = pandas.DataFrame(rows, columns=headers)
```

514
Get Suggestions for a Search (API)

To get suggestions for a search, use the GetSearchSuggestions (p. 1095) API.

The following example for AWS SDK for Python (Boto 3) is a get_search_suggestions request for items containing "linear".

```python
search_suggestion_params= {
  "Resource": "TrainingJob",
  "SuggestionQuery": {
    "PropertyNameQuery": {
      "PropertyNameHint": "linear"
    }
  }
}
```

The following is an example response for a get_search_suggestions request.

```json
{
  'PropertyNameSuggestions': [
    {'PropertyName': 'hyperparameters.linear_init_method'},
    {'PropertyName': 'hyperparameters.linear_init_value'},
    {'PropertyName': 'hyperparameters.linear_init_sigma'},
    {'PropertyName': 'hyperparameters.linear_lr'},
    {'PropertyName': 'hyperparameters.linear_wd'}
  ]
}
```

After getting search suggestions, you can use one of the property names in a search.

Verify the Datasets Used by Your Training Jobs

You can use model tracking capability to verify which datasets were used in training, where holdout datasets were used, and other details about training jobs. For example, use model tracking capability to verify that a specific dataset was used in a training job for an audit or to verify compliance.

To check whether a specific dataset was used in a training job, you search for the URL to its location in Amazon Simple Storage Service (Amazon S3). Model tracking capability returns the training jobs that used the dataset that you specify. If your search doesn't return the dataset (the result is empty), the dataset wasn't used in a training job. An empty result confirms, for example, that a holdout dataset wasn't used.

Trace Model Lineage

You can use model tracking capability to get information about the lineage of training jobs and the model resources that were used for them, including the dataset, algorithm, hyperparameters, and metrics. For example, if you find that the performance of a hosted model has declined, you can review its training job and the resources it used to determine what's causing the problem. You can use the Amazon SageMaker console or the API to trace lineage.
• Trace Model Lineage (Console) (p. 516)
• Trace Model Lineage (API) (p. 516)

Trace Model Lineage (Console)

To trace a model’s lineage (console)

2. In the navigation pane, choose Endpoints, and choose the relevant endpoint.
3. Scroll to the Endpoint configuration settings section. This section lists all of the model versions deployed at the endpoint, with a hyperlink to the training job that created each.

Trace Model Lineage (API)

To trace a model’s lineage, get the model’s name, then use it to search for training jobs.

The following example shows how to trace a model’s lineage using the API.

```python
# Get the name of model deployed at endpoint
endpoint_config = smclient.describe_endpoint_config(EndpointConfigName=endpointName)
model_name = endpoint_config['ProductionVariants'][0]['ModelName']

# Get the model's name
model = smclient.describe_model(ModelName=model_name)

# Search the training job by the location of model artifacts in Amazon S3
search_params={
    "MaxResults": 1,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "ModelArtifacts.S3ModelArtifacts",
                "Operator": "Equals",
                "Value": model['PrimaryContainer']['ModelDataUrl']
            }
        ]
    },
}
results = smclient.search(**search_params)

After finding the training job, you can review the resources used to train the model.

Amazon SageMaker Debugger

Amazon SageMaker Debugger provides full visibility into the training of machine learning models by monitoring, recording, and analyzing the tensor data that captures the state of a machine learning training job at each instance in its lifecycle. It provides a rich set of alerts when detecting errors for the steps of a machine learning training trial, and the ability to perform interactive explorations. It can automatically detect and alert you to commonly occurring errors such as gradient values getting too large or too small. When starting a training job that uses Debugger, you configure what tensors to save, where to save the tensors, and the trials to run on your dataset. The data collected remains in your account to use in subsequent analyses, securing its use for the most privacy-sensitive applications. Overall, Debugger can reduce the time to debug the training of models dramatically.

You can use Amazon SageMaker Debugger from Amazon SageMaker Studio or from Amazon SageMaker notebooks. Studio makes the inspection of training job issues easier by providing a visual interface for
you to use when analyzing your monitoring tensor data. You can use the Amazon SageMaker Debugger SDK to instrument your code to save tensors if necessary and to build customized rules. You can then use the Amazon SageMaker Python SDK to configure the debugger to save the required tensors and deploy built-in or custom rules monitoring these tensors. The open source smdebug Python library at Amazon SageMaker Debugger implements its core debugging functionality. Debugger also provides the parameters needed to hook up the smdebug functionality into the Amazon SageMaker training job APIs that enable it to be deployed on our machine learning platform.

Using Debugger is a two-step process:

- **Saving tensors (and scalars):** In deep learning algorithms, tensors define the state of the training job at any particular instant in its lifecycle. Amazon SageMaker Debugger exposes a library that allows you to capture these tensors and save them for analysis. Debugger is highly customizable and can help provide interoperability by saving performance and other feature-related metrics at specified frequencies.

- **Analysis:** The analysis of the tensors emitted is captured by the Amazon SageMaker Debugger concept called rules. On a very broad level, a rule is Python code used to detect certain conditions during training. Some of the conditions that a data scientist training an algorithm may care about are monitoring for gradients getting too large or too small or detecting overfitting. Debugger comes pre-packaged with certain rules. Users can write their own rules using the Debugger APIs. You can also analyze raw tensor data outside of the rules construct in an Amazon SageMaker notebook, using Debugger’s complete set of APIs.

**Amazon SageMaker Debugger Sample Notebooks**

Amazon SageMaker Debugger provides sample notebooks that show how to use four learning frameworks to emit tensors in an Amazon SageMaker training job and then how to apply rules over the tensors to monitor the status of these jobs. It also provides sample notebooks that you can run interactive analysis with using the open source smdebug Python library at Amazon SageMaker Debugger. The following notebooks are listed in the order we recommend you review them.

- Using a built-in rule with TensorFlow
- Using a custom rule with TensorFlow Keras
- Interactive tensor analysis in notebook with MXNet
- Visualizing Debugging Tensors of MXNet training
- Real-time analysis in notebook with MXNet
- Using a built in rule with XGBoost
- Real-time analysis in notebook with XGBoost
- Using SageMaker Debugger with Managed Spot Training and MXNet
- Reacting to CloudWatch Events from Rules to take an action based on status with TensorFlow
- Using SageMaker Debugger with a custom PyTorch container

The links to the Debugger sample notebooks is also available at Amazon SageMaker Debugger Examples.

**Note**

Although each of these notebooks focuses on a specific framework, the same approach works with all the other frameworks that Amazon SageMaker Debugger supports.

**Learning Frameworks and Built-in Algorithms Supported by Amazon SageMaker Debugger**

Amazon SageMaker Debugger supports using the following learning frameworks:
• Apache MXNet: Use Apache MXNet with Amazon SageMaker (p. 453)
• PyTorch: Use PyTorch with Amazon SageMaker (p. 454)
• TensorFlow: Use TensorFlow with Amazon SageMaker (p. 452)
• XGBoost: XGBoost Algorithm (p. 422)

Amazon SageMaker Debugger supports the following Amazon SageMaker built-in algorithm:

• XGBoost Release 0.72 (p. 434)—a supervised learning algorithm that is an open-source implementation of the gradient boosted trees algorithm.

There are two ways to enable Debugger while training models on Amazon SageMaker:

• Use framework containers we provide that do not require any script changes.
• Bring your own container and make the script changes needed to enable Debugger

Topics
• Zero Script Change (p. 518)
• Bring Your Own Training Container (p. 518)

Zero Script Change

We have provided custom versions of the framework containers we support that enable you to use Amazon SageMaker Debugger with no changes to your training script, by automatically adding Debugger’s Hook. We support the following frameworks and versions for this experience.

Supported Frameworks and Versions

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>1.15</td>
</tr>
<tr>
<td>MXNet</td>
<td>1.6</td>
</tr>
<tr>
<td>PyTorch</td>
<td>1.3</td>
</tr>
<tr>
<td>XGBoost</td>
<td>&gt;=0.90 (As built-in algorithm)</td>
</tr>
</tbody>
</table>

For more information on the deep learning frameworks containers, see Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch (p. 470).

Bring Your Own Training Container

This library smdebug itself supports versions other than the ones listed above. If you want to use SageMaker Debugger with a version different from the above, you will have to orchestrate your training script with a few lines. Before we discuss how these changes look like, let us take a look at the versions supported.

Supported Frameworks and Versions

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>1.13, 1.14, 1.15</td>
</tr>
</tbody>
</table>
To set up Debugger with your script on your container:

- Ensure that you are using Python 3+ runtime as smdebug only supports Python 3 or higher.
- Install `smdebug` binary through `pip install smdebug`
- Make some minimal modifications to your training script to add the hook for SageMaker Debugger. Refer to the framework pages linked in the table for instructions.

## Amazon CloudWatch Metrics for Amazon SageMaker Debugger

Amazon CloudWatch collects model training metrics so you can monitor training jobs. See the [Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics](p. 712) sections in the [Monitor Amazon SageMaker with Amazon CloudWatch](p. 712) topic.

### Topics
- [How Debugger Works (p. 519)]
- [Save Tensor Data for Debugger (p. 521)]
- [Prebuilt Amazon SageMaker Docker Images for Rules (p. 523)]
- [Built-in Rules Provided by Amazon SageMaker Debugger (p. 525)]
- [How to Use Built-in Rules for Model Analysis (p. 548)]
- [Programming Model for Debugger (p. 549)]
- [How to Use Custom Rules (p. 550)]
- [Amazon SageMaker Studio Visualizations of Model Analysis Results (p. 552)]
- [Amazon SageMaker Debugger Reference Documentation (p. 553)]

### How Debugger Works

Amazon SageMaker Debugger enables you go beyond just looking at scalars like losses and accuracies when evaluating model training. It gives you full visibility into a training job by using a hook to capture tensors that define the state of the training process at each instance in its lifecycle. It also provides the capability of defining 'rules' to analyze the captured tensors. Built-in rules monitor the training flow and alert you to problems with various common conditions that are critical for the success of the training job. You can also create your own custom rules to watch for any issues specific to your model. You can monitor the results of the analysis done by rules with Amazon CloudWatch events, using an Amazon SageMaker notebook, or in visualization provided by Amazon SageMaker Studio. Here is a depiction of the flow for the model training process with Amazon SageMaker Debugger .

The following diagram shows the flow for the model training process with Debugger.
Amazon SageMaker Debugger automatically debugs your machine learning training process. It helps you develop better, faster, cheaper models by catching common errors quickly. While training, it automatically saves tensors and the network state (no script changes needed). By constantly monitoring the training process, a separate job detects anomalies, such as a vanishing gradient, poor weight initialization, or other warning signals. That way, if a training job fails, you’ll know what happened and how to fix it. There are over 15 built-in “rules,” or state assertions to ensure that training happens efficiently.

Amazon SageMaker Debugger supports all popular machine learning frameworks (TensorFlow, PyTorch, and Apache MXNet) and XGBoost. It provides both an automated and a configurable experience. In the automated experience, it provides custom framework forks in the deep learning containers that automatically detect your training job and save tensors, without any changes to your training script. There is also an advanced configurable mode where you choose precisely which tensors to save, how to organize them, and which custom rules to use.

The data can be saved to an S3 bucket so you can run your own analysis afterwards by creating a “trial.” This provides full insight into the training process.

Amazon SageMaker Debugger does the following:

• Creates a “hook” object.
• Passes this hook as a callback inside the training process.
• The hook listens to various events, such as the forward and backward pass through the network. Upon registering a “step,” or forward and backward pass, it writes tensors and other data to your S3 bucket.
• The hook can also detect when your mode has switched between training and validation, letting you easily segment your data.
• The tensors are first saved locally on the training instance and then moved to the S3 location you specified.
• A separate instance runs the rule monitoring job which fetches tensors from the S3 location locally and invokes the rule logic on the tensors saved for the training job. If something is amiss, it raises an exception and triggers an Amazon CloudWatch event. You can run multiple rules simultaneously.
Save Tensor Data for Debugger

Tensors define the state of the training job at any particular instant in its lifecycle. Amazon SageMaker Debugger provides the smdebug library, which allows you to monitor these tensors, save them, and analyze them to evaluate model training. Tensors can be grouped into collections to help manage them. Debugger gives you a powerful and flexible API to save the tensors you choose at the frequencies you want. These configurations are made available in the Amazon SageMaker Python SDK through the DebuggerHookConfig class.

Topics
- Save Built-in Party Collections (p. 521)
- Save Reductions for a Custom Collection (p. 521)
- Enable TensorBoard Summaries (p. 522)

Save Built-in Party Collections

Learn more about these first party collections Common API.

```python
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig

hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    hook_parameters={
        "save_interval": 100
    },
    collection_configs=[
        CollectionConfig("weights"),
        CollectionConfig("gradients"),
        CollectionConfig("losses"),
        CollectionConfig(
            name="biases",
            parameters={
                "save_interval": 10,
                "end_step": 500
            }
        )
    ]
)

import sagemaker as sm

sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    debugger_hook_config=hook_config
)

sagemaker_estimator.fit()
```

Save Reductions for a Custom Collection

You can define your collection of tensors. You can also choose to save certain reductions of tensors only instead of saving the full tensor. You may choose to do this to reduce the amount of data saved.

**Note**

When you save reductions, unless you pass the flag `save_raw_tensor`, only these reductions will be available for analysis. The raw tensor will not be saved.
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig

hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    collection_configs=[
        CollectionConfig(
            name="activations",
            parameters={
                "include_regex": "relu|tanh",
                "reductions": "mean, variance, max, abs_mean, abs_variance, abs_max"
            }
        )
    ]
)

import sagemaker as sm

sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    debugger_hook_config=hook_config
)
sagemaker_estimator.fit()

Enable TensorBoard Summaries

Amazon SageMaker Debugger can automatically generate TensorBoard scalar summaries, distributions and histograms for tensors saved. This can be enabled by passing a TensorBoardOutputConfig object when creating an Estimator as follows. You can also choose to disable or enable histograms specifically for different collections. By default a collection has save_histogram flag set to True. Note that scalar summaries are added to TensorBoard for all ScalarCollections and any scalar saved through hook.save_scalar. For more information on scalar collections and save_scalar method, see the Common API.

The following example saves weights and gradients as full tensors, and also saves the gradients as histograms and distributions to visualize in TensorBoard. These are saved to the location passed in TensorBoardOutputConfig object.

from sagemaker.debugger import DebuggerHookConfig, CollectionConfig, TensorBoardOutputConfig

hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    collection_configs=[
        CollectionConfig(
            name="weights",
            parameters={"save_histogram": False}),
        CollectionConfig(name="gradients")
    ]
)

tb_config = TensorBoardOutputConfig('s3://smdebug-dev-demo-pdx/mnist/tensorboard')

import sagemaker as sm

sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
)
Prebuilt Amazon SageMaker Docker Images for Rules

Amazon SageMaker provides two kinds of prebuilt Docker images for rules, one for evaluating Amazon SageMaker owned rules and other for evaluating your custom rules as Python source files.

If you're using the Amazon SageMaker Python SDK, Amazon SageMaker rules can be used with an estimator, without having to retrieve a Docker image. If you are not using the SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-built container. The Amazon SageMaker prebuilt Rule Docker images are stored in Amazon Elastic Container Registry (Amazon ECR).

To pull an image from an Amazon ECR repo (or to push an image to an Amazon ECR repo), use the full name registry address of the image. Amazon SageMaker uses the following URL patterns for the container image registry addresses.

\(<account_id>.dkr.ecr.<region>.amazonaws.com/<ECR repo name>:<tag>\)

Topics
- Amazon SageMaker Built-in Rules Registry Ids (p. 523)
- Amazon SageMaker Custom Rule Evaluator Registry Ids (p. 524)

Amazon SageMaker Built-in Rules Registry Ids

The following table itemizes the supported values for the URL components of the registry addresses for the images providing built-in rules.

**ECR Repository Name:** sagemaker-debugger-rules

**Tag:** latest

**Example full name registry address:**

904829902805.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rules:latest

**Table: Regions and Build-in Rules Registry IDs**

<table>
<thead>
<tr>
<th>Regions</th>
<th>Registry IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap-east-1</td>
<td>199566480951</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>430734990657</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>578805364391</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>904829902805</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>972752614525</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>184798709955</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>519511493484</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>482524230118</td>
</tr>
</tbody>
</table>
### Amazon SageMaker Custom Rule Evaluator Registry IDs

The following table itemizes the supported values for the URL components of the registry addresses for the images providing custom rule evaluators.

**ECR Repository Name:** sagemaker-debugger-rule-evaluator

**Tag:** latest

**Example full name registry address:**

552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rule-evaluator:latest

### Table: Regions and Custom Rule Evaluator Registry IDs

<table>
<thead>
<tr>
<th>Regions</th>
<th>Registry IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-north-1</td>
<td>314864569078</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>929884845733</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>250201462417</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>447278800020</td>
</tr>
<tr>
<td>me-south-1</td>
<td>986000313247</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>818342061345</td>
</tr>
<tr>
<td>us-east-1</td>
<td>503895931360</td>
</tr>
<tr>
<td>us-east-2</td>
<td>915447279597</td>
</tr>
<tr>
<td>us-west-1</td>
<td>685455198987</td>
</tr>
<tr>
<td>us-west-2</td>
<td>895741380848</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>645844755771</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>670969264625</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>326368420253</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>552407032007</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>631532610101</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>445670767460</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>105842248657</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>691764027602</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>091235270104</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>606966180310</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>074613877050</td>
</tr>
</tbody>
</table>
### Built-in Rules Provided by Amazon SageMaker Debugger

Use the built-in rules provided by Amazon SageMaker Debugger to analyze tensors emitted during the training of machine learning models. These rules monitor various common conditions that are critical for the success of a training job. There are four scopes of validity for the built-in rules.

#### Scopes of Validity for Built-in Rules

<table>
<thead>
<tr>
<th>Scope of Validity</th>
<th>Built-in Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep learning frameworks (TensorFlow, Apache MXNet, and PyTorch)</td>
<td>• DeadRelu</td>
</tr>
<tr>
<td></td>
<td>• ExplodingTensor</td>
</tr>
<tr>
<td></td>
<td>• PoorWeightInitialization</td>
</tr>
<tr>
<td></td>
<td>• SaturatedActivation</td>
</tr>
<tr>
<td></td>
<td>• VanishingGradient</td>
</tr>
<tr>
<td></td>
<td>• WeightUpdateRatio</td>
</tr>
<tr>
<td>Deep learning frameworks (TensorFlow, MXNet, and PyTorch) and the XGBoost algorithm</td>
<td>• AllZero</td>
</tr>
<tr>
<td></td>
<td>• ClassImbalance</td>
</tr>
<tr>
<td></td>
<td>• Confusion</td>
</tr>
<tr>
<td></td>
<td>• LossNotDecreasing</td>
</tr>
<tr>
<td></td>
<td>• Overfit</td>
</tr>
<tr>
<td></td>
<td>• Overtraining</td>
</tr>
<tr>
<td></td>
<td>• SimilarAcrossRuns</td>
</tr>
<tr>
<td></td>
<td>• TensorVariance</td>
</tr>
<tr>
<td></td>
<td>• UnchangedTensor</td>
</tr>
<tr>
<td>Deep learning applications</td>
<td>• CheckInputImages</td>
</tr>
<tr>
<td></td>
<td>• NLPSequenceRatio</td>
</tr>
<tr>
<td>XGBoost algorithm</td>
<td>• TreeDepth</td>
</tr>
</tbody>
</table>

### Topics

- [DeadRelu Rule](p. 526)
- [ExplodingTensor Rule](p. 527)
DeadRelu Rule

This rule detects when the percentage of rectified linear unit (ReLU) activation functions in a trial are considered dead because their activation activity has dropped below a threshold. If the percent of inactive ReLUs in a layer is greater than the threshold_inactivity value of inactive ReLUs, the rule returns True.

Parameter Descriptions for the DeadRelu Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td>tensor_regex</td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: List of strings or a comma-separated string</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>threshold_inactivity</td>
<td>Defines a level of activity below which a ReLU is considered to be dead. A ReLU might be active in the beginning of a trial and then slowly die during the training process. If the ReLU is active less than</td>
</tr>
</tbody>
</table>
### Parameter Name
- **threshold_inactivity**: It is considered to be dead.  
  **Optional**  
  Valid values: Float  
  Default values: 1.0

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| threshold_layer | Returns True if the percentage of inactive ReLUs in a layer is greater than threshold_layer.  
  Returns False if the percentage of inactive ReLUs in a layer is less than threshold_layer.  
  **Optional**  
  Valid values: Float  
  Default values: 50.0 |

For an example of to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Note**  
This rule can't be applied to the XGBoost algorithm.

### ExplodingTensor Rule

This rule detects whether the tensors emitted during training have non-finite values, either infinite or NaN (not a number). If a non-finite value is detected, the rule returns True.

#### Parameter Descriptions for the ExplodingTensor Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| base_trial            | The trial run using this rule. The rule inspects the tensors gathered from this trial.  
  **Required**  
  Valid values: String |
| collection_names      | The list of collection names whose tensors the rule inspects.  
  **Optional**  
  Valid values: List of strings or a comma-separated string  
  Default value: None |
| tensor_regex          | A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors.  
  The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered |
### Parameter Descriptions for the PoorWeightInitialization Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td>activation_inputs_regex</td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match</td>
</tr>
<tr>
<td></td>
<td>the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
</tbody>
</table>

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis](#).  

**Note**  
This rule can't be applied to the XGBoost algorithm.

### PoorWeightInitialization Rule

This rule detects if your model parameters have been poorly initialized.

Good initialization breaks the symmetry of the weights and gradients in a neural network and maintains commensurate activation variances across layers. Otherwise, the neural network doesn't learn effectively. Initializers like Xavier aim to keep variance constant across activations, which is especially relevant for training very deep neural nets. Too small an initialization can lead to vanishing gradients. Too large an initialization can lead to exploding gradients. This rule checks the variance of activation inputs across layers, the distribution of gradients, and the loss convergence for the initial steps to determine if a neural network has been poorly initialized.
### Parameter Name | Description
---|---
| Valid values: List of strings or a comma-separated string | Default value: ".*relu_input"
| threshold | If the ratio between minimum and maximum variance of weights per layer exceeds the threshold at a step, the rule returns True. **Optional**
| Valid values: Float | Default value: 10.0
| distribution_range | If the minimum difference between 5th and 95th percentiles of the gradient distribution is less than the distribution range, the rule returns True. **Optional**
| Valid values: Float | Default value: 0.001
| patience | The number of steps to wait until the loss is considered to be no longer decreasing. **Optional**
| Valid values: Integer | Default value: 5
| steps | The number of steps this rule analyzes. You typically want to check only the first few iterations. **Optional**
| Valid values: Float | Default value: 10

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis](p. 548).

**Note**
This rule can't be applied to the XGBoost algorithm.

### SaturatedActivation Rule

This rule detects if the tanh and sigmoid activation layers are becoming saturated. An activation layer is saturated when the input of the layer is close to the maximum or minimum of the activation function. The minimum and maximum of the tanh and sigmoid activation functions are defined by their respective min_threshold and max_thresholds values. If the activity of a node drops below the
threshold_inactivity percentage, it is considered saturated. If more than a threshold_layer percent of the nodes are saturated, the rule returns True.

**Parameter Descriptions for the SaturatedActivation Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td>Valid values: String</td>
<td></td>
</tr>
</tbody>
</table>

| collection_names     | The list of collection names whose tensors the rule inspects.                                                                                                                                               |
| Valid values: List of strings or a comma-separated string |
| Default value: None   |

| tensor_regex         | A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched. |
| Valid values: List of strings or a comma-separated string |
| Default value: ".*tanh_input, .*sigmoid_input". |

| threshold_tanh       | The minimum and maximum thresholds that define the extremes of the input for a tanh activation function, defined as: (min_threshold, max_threshold). The default values are determined based on a vanishing gradient threshold of 0.0000001. |
| Valid values: Float  |
| Default values: (-9.4999, 9.4999) |

| threshold_sigmoid    | The minimum and maximum thresholds that define the extremes of the input for a sigmoid activation function, defined as: (min_threshold, max_threshold). The default values are determined based on a vanishing gradient threshold of 0.0000001. |
For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Note**

This rule can't be applied to the XGBoost algorithm.

### VanishingGradient Rule

This rule detects if the gradients in a trial become extremely small or drop to a zero magnitude. If the mean of the absolute values of the gradients drops below a specified threshold, the rule returns True.

#### Parameters Descriptions for the VanishingGradient Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. Required</td>
</tr>
<tr>
<td>threshold</td>
<td>The value at which the gradient is determined to be vanishing. Optional</td>
</tr>
</tbody>
</table>
### WeightUpdateRatio Rule

This rule keeps track of the ratio of updates to weights during training and detects if that ratio gets too large or too small. If the ratio of updates to weights is larger than the `large_threshold` value or if this ratio is smaller than `small_threshold`, the rule returns `True`.

Conditions for training are best when the updates are commensurate to the gradients. Excessively large updates can push the weights away from optimal values, and very small updates result in very slow convergence. This rule requires weights to be available for two consecutive steps, so `save_interval` needs to be set to 1.

**Parameter Descriptions for the WeightUpdateRatio Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td>num_steps</td>
<td>The number of steps across which the rule checks to determine if the tensor has changed.</td>
</tr>
<tr>
<td></td>
<td>The number of steps across which you want to compare the weight ratios. If you pass no value, the rule runs by default against the current step and the immediately previous saved step. If you override the default by passing a value for this parameter, the comparison is done between weights at step <code>s</code> and at a step that is <code>&gt;= s - num_steps</code>. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>large_threshold</td>
<td>The maximum value that the ratio of updates to weight can take before the rule returns <code>True</code>. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
</tbody>
</table>

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Note**

This rule can't be applied to the XGBoost algorithm.
### Built-in Rules

<table>
<thead>
<tr>
<th>Parameter Name,</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default value: 10.0</td>
</tr>
<tr>
<td>small_threshold</td>
<td>The minimum value that the ratio of updates to weigh can take, below which the rule returns True. <strong>Optional</strong> Valid values: Float Default value: 0.00000001</td>
</tr>
<tr>
<td>epsilon</td>
<td>A small constant used to ensure that Debugger does not divide by zero when computing the ratio updates to weigh. <strong>Optional</strong> Valid values: Float Default value: 0.000000001</td>
</tr>
</tbody>
</table>

**Note**
If tensors have been saved with TRAIN mode on during training, the rule runs only on TRAIN mode steps. Otherwise, it runs by default on GLOBAL mode steps.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis (p. 548)].

**Note**
This rule can't be applied to the XGBoost algorithm.

### AllZero Rule

This rule detects if all or a specified percentage of the values in the tensors are zero.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis (p. 548)].

#### Parameters Descriptions for the AllZero Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. <strong>Required</strong> Valid values: String</td>
</tr>
<tr>
<td>collection_names</td>
<td>The list of collection names whose tensors the rule inspects. <strong>Optional</strong></td>
</tr>
</tbody>
</table>
**Parameter Name** | **Description**  
---|---  
| tensor_regex | A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  
| Optional | Valid values: List of strings or a comma-separated string  
| | Default value: None  
| threshold | Specifies the percentage of values in the tensor that needs to be zero for this rule to be invoked.  
| Optional |  
| | Valid values: Float  
| | Default value: 100  

**ClassImbalance Rule**

This rule measures sampling imbalances between classes and throws errors if the imbalance exceeds a threshold or if too many mispredictions for underrepresented classes occur as a result of the imbalance.

Classification models require well-balanced classes in the training dataset or a proper weighting/sampling of classes during training. The rule performs two checks:

- It counts the occurrences per class. If the ratio of number of samples between smallest and largest class is larger than the threshold_imbalance, an error is thrown.
- It checks the prediction accuracy per class. If resampling or weighting has not been correctly applied, then the model can reach high accuracy for the class with many training samples, but low accuracy for the classes with few training samples. If a fraction of mispredictions for a certain class is above threshold_misprediction, an error is thrown.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Parameter Descriptions for the ClassImbalance Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>threshold_imbalance</td>
<td>The acceptable imbalance between the number of samples in the smallest class and in the largest class. Exceeding this threshold value throws an error.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>threshold_misprediction</td>
<td>A limit on the fraction of mispredictions allowed for each class. Exceeding this threshold throws an error. The underrepresented classes are most at risk of crossing this threshold.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.7</td>
</tr>
<tr>
<td>samples</td>
<td>The number of labels that have to be processed before an imbalance is evaluated. The rule might not be triggered until it has seen sufficient samples across several steps. The more classes that your dataset contains, the larger this sample number should be.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 500 (assuming a dataset like MNIST with 10 classes)</td>
</tr>
<tr>
<td>argmax</td>
<td>If True, np.argmax is applied to the prediction tensor. Required when you have a vector of probabilities for each class. It is used to determine which class has the highest probability.</td>
</tr>
<tr>
<td></td>
<td>Conditional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Boolean</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>labels_regex</td>
<td>The name of the tensor that contains the labels.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;.*labels&quot;</td>
</tr>
</tbody>
</table>
### Confusion Rule

This rule evaluates the goodness of a confusion matrix for a classification problem.

It creates a matrix of size $\text{category\_no} \times \text{category\_no}$ and populates it with data coming from $(\text{labels}, \text{predictions})$ pairs. For each $(\text{labels}, \text{predictions})$ pair, the count in $\text{confusion[labels][predictions]}$ is incremented by 1. When the matrix is fully populated, the ratio of data on-diagonal values and off-diagonal values are evaluated as follows:

- For elements on the diagonal: $\text{confusion[i][i]} / \text{sum}_j(\text{confusion[j][j]}) \geq \text{min\_diag}$
- For elements off the diagonal: $\text{confusion[j][i]} / \text{sum}_j(\text{confusion[j][i]}) \leq \text{max\_off\_diag}$

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

### Parameter Descriptions for the Confusion Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td>category_no</td>
<td>The number of categories.</td>
</tr>
<tr>
<td>labels</td>
<td>The tensor name for 1-d vector of true labels.</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>predictions</strong></td>
<td>The tensor name for 1-d vector of estimated labels.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;predictions&quot;</td>
</tr>
<tr>
<td><strong>labels_collection</strong></td>
<td>The rule inspects the tensors in this collection for labels.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;labels&quot;</td>
</tr>
<tr>
<td><strong>predictions_collection</strong></td>
<td>The rule inspects the tensors in this collection for predictions.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;predictions&quot;</td>
</tr>
<tr>
<td><strong>min_diag</strong></td>
<td>The minimum value for the ratio of data on the diagonal.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{float} \leq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td><strong>max_off_diag</strong></td>
<td>The maximum value for the ratio of data off the diagonal.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{float} \leq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
</tbody>
</table>

**Note**
This rule infers default values for the optional parameters if their values aren't specified.

**LossNotDecreasing Rule**

This rule detects when the loss is not decreasing in value at an adequate rate. These losses must be scalars.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.
For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

### Parameter Descriptions for the LossNotDecreasing Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base_trial</strong></td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td><strong>collection_names</strong></td>
<td>The list of collection names whose tensors the rule inspects.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td><strong>tensor_regex</strong></td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td><strong>use_losses_collection</strong></td>
<td>If set to True, looks for losses in the collection named “losses” when the collection is present.</td>
</tr>
<tr>
<td><strong>nym_steps</strong></td>
<td>The minimum number of steps after which the rule checks if the loss has decreased. Rule evaluation happens every <strong>num_steps</strong>. The rule compares the loss for this step with the loss at a step which is at least <strong>num_steps</strong> behind the current step. For example, suppose that the loss is being saved every 3 steps, but <strong>num_steps</strong> is set to 10. At step 21, loss for step 21 is compared with the loss for step 9. The next step where loss is checked is 33, because 10 steps after 21 is 31, and at 31 and 32 loss is not being saved.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Valid values:</strong> Integer</td>
<td></td>
</tr>
<tr>
<td><strong>Default value:</strong> 10</td>
<td></td>
</tr>
<tr>
<td><strong>diff_percent</strong></td>
<td>The minimum percentage difference by which the loss should decrease between <code>num_steps</code>.</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Valid values:</strong> <code>0.0 &lt; float &lt; 100</code></td>
<td></td>
</tr>
<tr>
<td><strong>Default value:</strong> Checks if the loss is decreasing between <code>num_steps</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>The name of the Debugger mode to query tensor values for rule checking. If this is not passed, the rule checks in order by default for the mode.EVAL, then mode.TRAIN, and then mode.GLOBAL.</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Valid values:</strong> String (EVAL, TRAIN, OR GLOBAL)</td>
<td></td>
</tr>
<tr>
<td><strong>Default value:</strong> None</td>
<td></td>
</tr>
</tbody>
</table>

**Overfit Rule**

This rule detects if your model is being overfit to the training data by comparing the validation and training losses.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Note**

A standard way to prevent overfitting is to regularize your model.

**Parameter Descriptions for the Overfit Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base_trial</strong></td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td><strong>Required</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Valid values:</strong> String</td>
<td></td>
</tr>
<tr>
<td><strong>tensor_regex</strong></td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
in the trials by default. Only scalar-valued tensors can be matched.
Optional
Valid values: List of strings or a comma-separated string
Default value: None

| start_step | The step from which to start comparing the validation and training loss.
Optional
Valid values: Integer
Default value: 0

| patience | The number of steps for which the ratio_threshold is allowed to exceed the value set before the model is considered to be overfit.
Optional
Valid values: Integer
Default value: 1

| ratio_threshold | The maximum ratio of the difference between the mean validation loss and mean training loss to the mean training loss. If this threshold is exceeded for a patience number of steps, the model is being overfit and the rule returns True.
Optional
Valid values: Float
Default value: 0.1

## Overtraining Rule
This rule detects if the model is being overtrained.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

### Note
Overtraining can be avoided by early stopping. For information on early stopping, see Stop Training Jobs Early (p. 568). For an example that shows how to use spot training with Debugger, see Enable Spot Training with Amazon SageMaker Debugger.
Parameter Descriptions for the Overtraining Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td>patience_train</td>
<td>The number of steps to wait before the training loss is considered to not to be improving anymore.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>patience_validation</td>
<td>The number of steps to wait before the validation loss is considered to not to be improving anymore.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>delta</td>
<td>The minimum threshold by how much the error should improve before it is considered as a new optimum.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
</tbody>
</table>

The following Python example shows how to implement this rule.

```python
rules_specification = [
    {
        "RuleName": "Overtraining",
        "InstanceType": "ml.c5.4xlarge",
        "RuntimeConfigurations": {
            "patience_train": "10",
            "patience_validation": "20"
        }
    }
]
```

SimilarAcrossRuns Rule

This rule compares tensors gathered from a base trial with tensors from another trial.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.
For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Parameter Descriptions for the SimilarAcrossRuns Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. Required</td>
</tr>
<tr>
<td>other_trial</td>
<td>The trial whose tensors you want to compare to those tensors gathered from the base_trial. Required</td>
</tr>
<tr>
<td>collection_names</td>
<td>The list of collection names whose tensors the rule inspects. Optional</td>
</tr>
<tr>
<td>tensor_regex</td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched. Optional</td>
</tr>
</tbody>
</table>

**TensorVariance Rule**

This rule detects if you are having tensors with very high or low variances. Very high or low variances in a tensor could lead to neuron saturation, which reduces the learning ability of the neural network. Very high variance in tensors can also eventually lead to exploding tensors. Use this rule to detect such issues early.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the collection_names or tensor_regex parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.
For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

### Parameter Descriptions for the TensorVariance Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base_trial</strong></td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. Required&lt;br&gt;Required&lt;br&gt;Valid values: String</td>
</tr>
<tr>
<td><strong>collection_names</strong></td>
<td>The list of collection names whose tensors the rule inspects. Optional&lt;br&gt;Optional&lt;br&gt;Valid values: List of strings or a comma-separated string&lt;br&gt;Default value: None</td>
</tr>
<tr>
<td><strong>tensor_regex</strong></td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched. Optional&lt;br&gt;Optional&lt;br&gt;Valid values: List of strings or a comma-separated string&lt;br&gt;Default value: None</td>
</tr>
<tr>
<td><strong>max_threshold</strong></td>
<td>The threshold for the upper bound of tensor variance. Optional&lt;br&gt;Optional&lt;br&gt;Valid values: Float&lt;br&gt;Default value: xxx</td>
</tr>
<tr>
<td><strong>min_threshold</strong></td>
<td>The threshold for the lower bound of tensor variance. Optional&lt;br&gt;Optional&lt;br&gt;Valid values: Float&lt;br&gt;Default value: xxx</td>
</tr>
</tbody>
</table>
UnchangedTensor Rule

This rule detects whether a tensor is no longer changing across steps.

This rule runs the `numpy.allclose` method to check if the tensor isn’t changing.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

Parameter Descriptions for the UnchangedTensor Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String</td>
</tr>
<tr>
<td>collection_names</td>
<td>The list of collection names whose tensors the rule inspects.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: List of strings or a comma-separated string</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>tensor_regex</td>
<td>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: List of strings or a comma-separated string</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>num_steps</td>
<td>The number of steps across which the rule checks to determine if the tensor has changed.</td>
</tr>
<tr>
<td></td>
<td>This checks the last <code>num_steps</code> that are available. They don’t need to be consecutive. If <code>num_steps</code> is 2, at step s it doesn’t necessarily check for s-1 and s. If s-1 isn’t available, it checks the last available step along with s. In that case, it checks the last available step with the current step.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
</tbody>
</table>
### Built-in Rules

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rtol</strong></td>
<td>The relative tolerance parameter to be passed to the <code>numpy.allclose</code> method. <strong>Optional</strong> Valid values: Float Default value: 1e-05</td>
</tr>
<tr>
<td><strong>atol</strong></td>
<td>The absolute tolerance parameter to be passed to the <code>numpy.allclose</code> method. <strong>Optional</strong> Valid values: Float Default value: 1e-08</td>
</tr>
<tr>
<td><strong>equal_nan</strong></td>
<td>Whether to compare NaNs as equal. If True, NaNs in input array a are considered equal to NaNs in input array b in the output array. This parameter is passed to the <code>numpy.allclose</code> method. <strong>Optional</strong> Valid values: Boolean Default value: False</td>
</tr>
</tbody>
</table>

---

**CheckInputImages Rule**

This rule checks if input images have been correctly normalized. Specifically, it detects if the mean of the sample data differs by more than a threshold value from zero. Many computer vision models require that input data has a zero mean and unit variance.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Parameter Descriptions for the CheckInputImages Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base_trial</strong></td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial. <strong>Required</strong> Valid values: String</td>
</tr>
<tr>
<td><strong>threshold_mean</strong></td>
<td>A threshold that defines by how much mean of the input data can differ from 0.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>threshold_samples</td>
<td>The number of images that have to be sampled before an error can be thrown. If the value is too low, the estimation of the dataset mean will be inaccurate. Optional Valid values: Integer Default value: 500</td>
</tr>
<tr>
<td>regex</td>
<td>The name of the input data tensor. Optional Valid values: String Default value: &quot;.*hybridsequential0_input_0&quot; (the name of the input tensor for Apache MXNet models using HybridSequential)</td>
</tr>
<tr>
<td>channel</td>
<td>The position of the color channel in the input tensor. Optional Valid values: Integer Default value: 1 (for example, MXNet expects input data in the form of (batch_size, channel, height, width))</td>
</tr>
</tbody>
</table>

**NLPSequenceRatio Rule**

This rule calculates the ratio of specific tokens given the rest of the input sequence that is useful for optimizing performance. For example, you can calculate the percentage of padding end-of-sentence (EOS) tokens in your input sequence. If the number of EOS tokens is too high, an alternate bucketing strategy should be performed. You also can calculate the percentage of unknown tokens in your input sequence. If the number of unknown words is too high, an alternate vocabulary could be used.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

**Parameter Descriptions for the NLPSequenceRatio Rule**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
</tbody>
</table>
### Parameter Name

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
</tr>
<tr>
<td>Valid values: String</td>
</tr>
</tbody>
</table>

### tensor_regex

- **A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors.** The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.
- **Optional**
- **Valid values:** List of strings or a comma-separated string
- **Default value:** ".*embedding0_input_0" (assuming an embedding as the initial layer of the network)

### token_values

- **A string of a list of the numerical values of the tokens.** For example, "3, 0".
- **Optional**
- **Valid values:** Comma-separated string of numerical values
- **Default value:** 0

### token_thresholds_percent

- **A string of a list of thresholds (in percentages) that correspond to each of the token_values.** For example, "50.0, 50.0".
- **Optional**
- **Valid values:** Comma-separated string of floats.
- **Default value:** "50, 50"

---

### TreeDepth Rule

This rule measures the depth of trees in an XGBoost model. XGBoost rejects splits if they don’t improve loss. This regularizes the training. As a result, the tree might not grow as deep as defined in `max_depth`.

This rule is valid only for the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see How to Use Built-in Rules for Model Analysis (p. 548).

### Parameter Descriptions for the TreeDepth Rule

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_trial</td>
<td>The trial run using this rule. The rule inspects the tensors gathered from this trial.</td>
</tr>
</tbody>
</table>
### Use Built-in Rules

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>The depth of the tree. The depth of the tree is obtained by computing the base 2 logarithm of the largest node ID.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Valid values: String</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: 4</td>
</tr>
</tbody>
</table>

#### How to Use Built-in Rules for Model Analysis

Amazon SageMaker Debugger rules analyse tensors emitted during the training of a model. They monitor conditions that are critical for the success of a training job. For example, they can detect whether gradients are getting too large or too small or if a model is being overfit. Debugger comes pre-packaged with certain Python-coded, built-in rules.

You can deploy a built-in rule to monitor your training job either by using the `CreateTrainingJob (p. 931)` API or by using the open source `smdebug Python library` with the Amazon SageMaker Python SDK. The `smdebug` programming model provides the context for understanding this task. For information on the programming model, see Analysis.

**Note**

You are not charged for the instances when running SageMaker built-in rules.

**Topics**

- Use the CreateTrainingJob API to Create a Built-in Rule (p. 548)
- Use smdebug Python library with the Amazon SageMaker Python SDK to Create a Built-in Rule (p. 549)

#### Use the CreateTrainingJob API to Create a Built-in Rule

A built-in rule can be configured for a training job using the `DebugHookConfig (p. 1331)` and `DebugRuleConfiguration (p. 1333)` objects in the `CreateTrainingJob (p. 931)` API. These rules are run on one of our pre-built Docker images which are listed in the Prebuilt Amazon SageMaker Docker Images for Rules (p. 523) topic. You specify the URL registry address for the pre-built Docker image in the `RuleEvaluatorImage` parameter.

The following code sample shows how to configure a built-in VanishingGradient rule using this Amazon SageMaker API.

```python
DebugRuleConfigurations: [
    {
        "RuleConfigurationName": "Amazon-VanishingGradient",
        "RuleEvaluatorImage": "503895931360.dkr.ecr.us-east-1.amazonaws.com/sagemaker-debugger-rules:latest",
        "RuleParameters": {
            "rule_to_invoke": "VanishingGradient",
            "threshold": "20.0"
        }
    }
]```
With a configuration like the one in this sample, Amazon SageMaker Debugger starts a rule evaluation job for your training job using the Amazon SageMaker VanishingGradient rule.

Use `smdebug` Python library with the Amazon SageMaker Python SDK to Create a Built-in Rule

For a sample notebook that shows you how to use a built-in rule when training job with a TensorFlow model in SM; see Amazon SageMaker Debugger - Using built-in rule.

The following code sample shows how to run an Amazon SageMaker built-in Rule using the `Rule.sagemaker` method from the Amazon SageMaker Python SDK. The first argument to this method is the base configuration that is associated with the Rule. We configure the built-in rules with the `smdebug_ruleconfigs` that we populate for all of the built-in rules. For details, see Amazon SageMaker Debugger `Rule.sagemaker`. We provide default values for the parameters that are not required, but you have the option to modify these default values when using the built-in rules. You do not need to specify a pre-built Docker image when using the Amazon SageMaker Python SDK as the Estimators handle this task.

```python
from sagemaker.debugger import Rule, CollectionConfig, rule_configs

exploding_tensor_rule = Rule.sagemaker(
    base_config=rule_configs.exploding_tensor(),
    rule_parameters={"collection_names": "weights,losses"},
    collections_to_save=[
        CollectionConfig("weights"),
        CollectionConfig("losses")
    ]
)

vanishing_gradient_rule = Rule.sagemaker(
    base_config=rule_configs.vanishing_gradient()
)

import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    rules=[exploding_tensor_rule, vanishing_gradient_rule],
)

sagemaker_estimator.fit()
```

You can use these rules or write your own rules using the Amazon SageMaker Debugger APIs. You can also analyze raw tensor data without using rules in, for example, an Amazon SageMaker notebook, using Debugger's full set of APIs.

Programming Model for Debugger

The programming model is organized around three main constructs that characterize Amazon SageMaker Debugger training jobs: trial, tensor, mode, and rules.

- **Trial**: A `smdebug_trial` is an object which lets you query for tensors for a given Debugger training job, specified by the path where smdebug's artifacts are saved. Trial is capable of loading new tensors as
and when they become available at the given path, allowing you to do both offline as well as realtime analysis. For information on methods in the smdebug Trial API, see Trial API.

- **Tensor**: Tensors define the state of the training job at any API particular instant in its lifecycle. An smdebug Tensor object can be retrieved through the `trial.tensor(name)` API. It is uniquely identified by the string representing its name. For information on methods in the smdebug Tensor API, see Tensor API. For information on how to enable or disable a refresh of tensors when querying a trial with smdebug.analysis.utils, see Utils.

- **Rule**: Debugger uses rules to execute a certain piece of code regularly on different steps of the jobs. A rule is assigned to a trial and can be invoked at each new step of the trial. You can evaluate a rule using tensors from the current step or any step before the current step. You must ensure that your logic respects these semantics, else you get a `TensorUnavailableForStep` exception as the data would not yet be available for future steps. You can use the build-in rules that Debugger provides or you can write your own custom rules. For information on how to write a custom rule, see How to Use Custom Rules (p. 550). For information on how to use built-in rules, see How to Use Built-in Rules for Model Analysis (p. 548).

For more information on the Amazon SageMaker Debugger programming model see Programming Model for Analysis

### How to Use Custom Rules

You can deploy a custom rule to monitor your training job either by using the CreateTrainingJob (p. 931) API or by using the open source smdebug Python library with the Amazon SageMaker Python SDK. The smdebug programming model provides the context for understanding this task. For information on the programming model, see Analysis.

To run a custom rule, you have to provide a few additional parameters for the interface. Key parameters are the Python file that has the implementation of your `Rule` class, the name of the `Rule` class, the type of instance to run the `Rule` job on, the size of the volume on that instance, and the docker image to use for running this job.

**Topics**

- Use the CreateTrainingJob API to Create a Custom Rule (p. 550)
- Use smdebug Python library with the Amazon SageMaker Python SDK to Create a Custom Rule (p. 551)

### Use the CreateTrainingJob API to Create a Custom Rule

A custom rule can be configured for a training job using the `DebugHookConfig` (p. 1331) and `DebugRuleConfiguration` (p. 1333) objects in the CreateTrainingJob (p. 931) API. The following code sample shows how to configure a custom `ImproperActivation` rule written with the smdebug library using this Amazon SageMaker API. This example assumes that you've written the custom rule in `custom_rules.py` file and uploaded it to an S3 bucket. We provide pre-built Docker images that you can use to run your custom rules. These are listed at Amazon SageMaker Custom Rule Evaluator Registry Ids (p. 524). You specify the URL registry address for the pre-built Docker image in the RuleEvaluatorImage parameter.

```json
DebugHookConfig: {
    "S3OutputPath": "s3://bucket/",
    "CollectionConfigurations": [ {
        "CustomCollection": ",
        "CollectionParameters": { 
            "include_regex": "relu",
            "save_interval": "500",
        
```
Use `smdebug` Python library with the Amazon SageMaker Python SDK to Create a Custom Rule

For a sample notebook that shows you how to use a custom rule to monitor your training job with a tf.keras ResNet example, see Amazon SageMaker - Debugging with custom rules.

The following code sample shows how to configure a custom `ImproperActivation` rule using the open source `smdebug` Python library with the Amazon SageMaker Python SDK. This example assumes that the custom rule you've written has path `/rules/custom_rules.py`. You do not need to specify a pre-built Docker image when using the Amazon SageMaker Python SDK as the Estimators handle this task.

```python
from sagemaker.debugger import Rule, CollectionConfig

custom_coll = CollectionConfig(
    name='relu_activations',
    parameters={
        "include_regex": "relu",
        "save_interval": "500",
        "end_step": "5000"
    })

improper_activation_rule = Rule.custom(
    name='improper_activation_job',
    image_uri='552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rule-evaluator:latest',
    instance_type='ml.c4.xlarge',
    volume_size_in_gb=400,
    rule_parameters={
        "source_s3_uri": "s3://bucket/custom_rules.py",
        "rule_to_invoke": "ImproperActivation",
        "collection_names": "relu_activations"
    },
    collections_to_save=[custom_coll]
)

import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    rules=[improper_activation_rule],
)
```
Amazon SageMaker Developer Guide
Visualize Model Analysis

Amazon SageMaker Studio Visualizations of Model Analysis Results

Amazon SageMaker Studio provides visualizations to interpret tensor outputs from model analysis.

Loss curves while training is in progress

The following screenshot shows visualizations of loss curves for training. The training is in progress.

Analyzing training jobs - comparing loss curves across multiple jobs

Amazon SageMaker Studio allows simple comparison across multiple jobs - in this case the loss. This helps identify the best-performing training jobs.
Rules triggering + logs from Debugger Jobs

When rules are triggered for anomalous conditions, Amazon SageMaker Studio presents logs for the failing rule, allowing easy analysis of the causes of the condition.

Amazon SageMaker Debugger Reference Documentation

The following sections contain reference documentation on the APIs, exceptions and some known limitations for Debugger

Topics

- Amazon SageMaker Debugger API (p. 553)
- Amazon SageMaker Debugger Exceptions (p. 554)
- Amazon SageMaker Debugger Best Practices (p. 554)
- Known Limitations with Amazon SageMaker Debugger (p. 555)

Amazon SageMaker Debugger API

Amazon SageMaker Debugger has APIs in several locations that are used to implement its monitoring and analysis of model training.

Amazon SageMaker Debugger provides an open source smdebug Python Library at awslabs/sagemaker-debugger that is used to configure built-in rules or to define custom rules used to analyze the tensor data from training jobs.

The Amazon SageMaker Python SDK is a high level SDK focused on machine learning experimentation. The SDK can be used to deploy built-in or custom rules defined with the smdebug Python Library to monitor and analyse these tensors using Amazon SageMaker estimators.

Debugger has added operations and types to the Amazon SageMaker API that enable the platform to use Debugger when training a model and to manage the configuration of inputs and outputs. Here is a list of the APIs to hook up Debugger.
CreateTrainingJob (p. 931) and DescribeTrainingJob (p. 1066) use the following types to configure tensor and rule configurations, hook up the smdebug library, and manage the storage of TensorBoard outputs:

- DebugRuleConfiguration (p. 1333)
- CollectionConfiguration (p. 1318)
- DebugHookConfig (p. 1331)
- TensorBoardOutputConfig (p. 1519)

DescribeTrainingJob (p. 1066) also provides an additional type to report on the status of rule evaluations:

- DebugRuleEvaluationStatus (p. 1335)

TrainingJob (p. 1520) also has parameters for accessing these Debugger configuration types as well as the time and billable time spent in model training.

Debugger also makes use of the Amazon SageMaker Processing functionality when analyzing model training. For more information on Processing, see Process Data and Evaluate Models (p. 189).

Amazon SageMaker Debugger Exceptions

Amazon SageMaker Debugger is designed to be aware that tensors required to execute a rule may not be available at every step. Hence it raises a few exceptions which allow us to control what happens when a tensor is missing. There are These are available in the smdebug.exceptions module. You can import them as follows:

```python
from smdebug.exceptions import *
```

Here are the exceptions and their meanings:

- TensorUnavailableForStep: The tensor requested is not available for the step. This might mean that this step might not be saved at all by the hook, or that this step might have saved some tensors but the requested tensor is not part of them. Note that when you see this exception, it means that this tensor can never become available for this step in the future. If the tensor has reductions saved for the step, it notifies you they can be queried.
- TensorUnavailable: This tensor is not being saved or has not been saved by the smdebug API. This means that this tensor will never be seen for any step in smdebug.
- StepUnavailable: The step was not saved and Debugger has no data from the step.
- StepNotYetAvailable: The step has not yet been seen by smdebug. It may be available in the future if the training is still going on. Debugger automatically loads new data as and when it becomes available.
- NoMoreData: Raised when the training ends. Once you see this, you know that there are no more steps and no more tensors to be saved.
- IndexReaderException: The index reader is not valid.
- InvalidWorker: A worker was invoked that was not valid.
- RuleEvaluationConditionMet: Evaluation of the rule at the step resulted in the condition being met.
- InsufficientInformationForRuleInvocation: Insufficient information was provided to invoke the rule.

Amazon SageMaker Debugger Best Practices

Step Intervals: By default Python SDK uses step interval as 500 for emitting tensors. If you want to emit more frequently, choose specific tensors and run for fewer epochs to avoid stressing the CPU/disk.
Known Limitations with Amazon SageMaker Debugger

Here are the known limitations for Amazon SageMaker Debugger:

- **TensorFlow support**: It does not support TensorFlow 2.0.
- **Horovod support**: It does not support training on jobs that use distributed training with the TensorFlow Horovod container. (Distributed training with Horovod is supported for MXNet and PyTorch.)
- **Distributed training**: Parameter server-based distributed training is not supported for MXNet and TensorFlow.

Perform Automatic Model Tuning

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a binary classification problem on a marketing dataset. Your goal is to maximize the area under the curve (auc) metric of the algorithm by training an XGBoost Algorithm (p. 422) model. You don't know which values of the eta, alpha, min_child_weight, and max_depth hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that Amazon SageMaker hyperparameter tuning searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

You can use Amazon SageMaker automatic model tuning with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for machine learning frameworks.

Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

You should also prepare your dataset and algorithm so that they work in Amazon SageMaker and successfully run a training job at least once. For information about setting up and running a training job, see Get Started with Amazon SageMaker (p. 20).

**Topics**

- How Hyperparameter Tuning Works (p. 556)
- Define Metrics (p. 557)
- Define Hyperparameter Ranges (p. 558)
- Example: Hyperparameter Tuning Job (p. 560)
- Stop Training Jobs Early (p. 568)
- Run a Warm Start Hyperparameter Tuning Job (p. 570)
- Automatic Model Tuning Resource Limits (p. 574)
- Best Practices for Hyperparameter Tuning (p. 574)
How Hyperparameter Tuning Works

Random Search

In a random search, hyperparameter tuning chooses a random combination of values from within the ranges that you specify for hyperparameters for each training job it launches. Because the choice of hyperparameter values doesn't depend on the results of previous training jobs, you can run the maximum number of concurrent training jobs without affecting the performance of the search.

For an example notebook that uses random search, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.

Bayesian Search

Bayesian search treats hyperparameter tuning like a [regression] problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Hyperparameter tuning uses an Amazon SageMaker implementation of Bayesian optimization.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

For more information about Bayesian optimization, see the following:

Basic Topics on Bayesian Optimization

- A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning
- Practical Bayesian Optimization of Machine Learning Algorithms
- Taking the Human Out of the Loop: A Review of Bayesian Optimization

Speeding up Bayesian Optimization

- Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization
- Google Vizier: A Service for Black-Box Optimization
- Learning Curve Prediction with Bayesian Neural Networks
- Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves

Advanced Modeling and Transfer Learning

- Scalable Hyperparameter Transfer Learning
- Bayesian Optimization with Tree-structured Dependencies
- Bayesian Optimization with Robust Bayesian Neural Networks
• Scalable Bayesian Optimization Using Deep Neural Networks
• Input Warping for Bayesian Optimization of Non-stationary Functions

Note
Hyperparameter tuning might not improve your model. It is an advanced tool for building machine solutions, and, as such, should be considered part of the scientific development process. When you build complex machine learning systems like deep learning neural networks, exploring all of the possible combinations is impractical. Hyperparameter tuning can accelerate your productivity by trying many variations of a model, focusing on the most promising combinations of hyperparameter values within the ranges that you specify. To get good results, you need to choose the right ranges to explore. Because the algorithm itself is stochastic, it’s possible that the hyperparameter tuning model will fail to converge on the best answer, even if the best possible combination of values is within the ranges that you choose.

Define Metrics

Note
When you use one of the Amazon SageMaker built-in algorithms, you don’t need to define metrics. Built-in algorithms automatically send metrics to hyperparameter tuning. You do need to choose one of the metrics that the built-in algorithm emits as the objective metric for the tuning job. For a list of metrics that a built-in algorithm emits, see the Metrics table for the algorithm in Use Amazon SageMaker Built-in Algorithms (p. 220).

To optimize hyperparameters for a machine learning model, a tuning job evaluates the training jobs it launches by using a metric that the training algorithm writes to logs. Amazon SageMaker hyperparameter tuning parses your algorithm’s stdout and stderr streams to find algorithm metrics, such as loss or validation-accuracy, that show how well the model is performing on the dataset.

Note
These are the same metrics that Amazon SageMaker sends to CloudWatch Logs. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 719).

If you use your own algorithm for hyperparameter tuning, make sure that your algorithm emits at least one metric by writing evaluation data to stderr or stdout.

Note
Hyperparameter tuning sends an additional hyperparameter, _tuning_objective_metric to the training algorithm. This hyperparameter specifies the objective metric being used for the hyperparameter tuning job, so that your algorithm can use that information during training.

You can define up to 20 metrics for your tuning job to monitor. You choose one of those metrics to be the objective metric, which hyperparameter tuning uses to evaluate the training jobs. The hyperparameter tuning job returns the training job that returned the best value for the objective metric as the best training job.

You define metrics for a tuning job by specifying a name and a regular expression for each metric that your tuning job monitors. Design the regular expressions to capture the values of metrics that your algorithm emits. You pass these metrics to the CreateHyperParameterTuningJob (p. 890) operation in the TrainingJobDefinition parameter as the MetricDefinitions field of the AlgorithmSpecification field.

The following example defines 4 metrics:

```json
{{
    "Name": "loss",
    "Regex": "Loss = (.*)",
```
To define hyperparameter ranges by using the low-level API, you specify the names of hyperparameters and ranges of values in the ParameterRanges field of the HyperParameterTuningJobConfig parameter that you pass to the CreateHyperParameterTuningJob (p. 890) operation. The ParameterRanges field has three subfields, one for each of the categorical, integer, and continuous hyperparameter ranges. You can define up to 20 hyperparameters to search over. Each value of a categorical hyperparameter range counts as a hyperparameter against the limit. Hyperparameter ranges have the following structure:

```
"ParameterRanges": {
  "CategoricalParameterRanges": [
    {
      "Name": "tree_method",
      "Values": ["auto", "exact", "approx", "hist"]
    }
  ],
  "ContinuousParameterRanges": [
    {
      "Name": "eta",
      "Range": [0.1, 1.0]
    }
  ]
}
```
Hyperparameter Scaling

For integer and continuous hyperparameter ranges, you can choose the scale you want hyperparameter tuning to use to search the range of values by specifying a value for the `ScalingType` field of the hyperparameter range. You can choose from the following scaling types:

Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale. Typically, you choose this if the range of all values from the lowest to the highest is relatively small (within one order of magnitude), because uniformly searching values from the range will give you a reasonable exploration of the entire range.

Logarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

Choose logarithmic scaling when you are searching a range that spans several orders of magnitude. For example, if you are tuning a Tune a Linear Learner Model (p. 328) model, and you specify a range of values between 0.0001 and 1.0 for the `learning_rate` hyperparameter, searching uniformly on a logarithmic scale gives you a better sample of the entire range than searching on a linear scale would, because searching on a linear scale would, on average, devote 90 percent of your training budget to only the values between 0.1 and 1.0, leaving only 10 percent of your training budget for the values between 0.001 and 0.1.

ReverseLogarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale. Reverse logarithmic scaling is supported only for continuous hyperparameter ranges. It is not supported for integer hyperparameter ranges.

Reverse logarithmic scaling works only for ranges that are entirely within the range $0 \leq x < 1.0$.

Choose reverse logarithmic scaling when you are searching a range that is highly sensitive to small changes that are very close to 1.

For an example notebook that uses hyperparameter scaling, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.
Example: Hyperparameter Tuning Job

This example shows how to create a new notebook for configuring and launching a hyperparameter tuning job. The tuning job uses the XGBoost Algorithm (p. 422) to train a model to predict whether a customer will enroll for a term deposit at a bank after being contacted by phone.

You use the low-level AWS SDK for Python (Boto) to configure and launch the hyperparameter tuning job, and the AWS Management Console to monitor the status of hyperparameter training jobs. You can also use the Amazon SageMaker high-level Amazon SageMaker Python SDK to configure, run, monitor, and analyze hyperparameter tuning jobs. For more information, see https://github.com/aws/sagemaker-python-sdk.

Prerequisites

To run the code in this example, you need

- An AWS account and an administrator user (p. 15)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training (p. 25)
- A running Amazon SageMaker notebook instance (p. 26)

Topics

- Create a Notebook (p. 560)
- Get the Amazon Sagemaker Boto 3 Client (p. 561)
- Get the Amazon SageMaker Execution Role (p. 561)
- Specify a Bucket and Data Output Location (p. 561)
- Download, Prepare, and Upload Training Data (p. 562)
- Configure and Launch a Hyperparameter Tuning Job (p. 563)
- Monitor the Progress of a Hyperparameter Tuning Job (p. 566)
- Clean up (p. 568)

Create a Notebook

Create a Jupyter notebook that contains a preinstalled environment with the default Anaconda installation and Python3.

To create a Jupyter notebook

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Open a running notebook instance, by choosing Open next to its name. The Jupyter notebook server page appears:

3. To create a notebook, choose Files, New, and conda_python3.
4. Name the notebook.

Next Step

Get the Amazon Sagemaker Boto 3 Client (p. 561)

Get the Amazon Sagemaker Boto 3 Client

Import libraries and get a Boto3 client, which you use to call the hyperparameter tuning APIs.

In the new Jupyter notebook, type the following code:

```python
import sagemaker
import boto3
from sagemaker.predictor import csv_serializer  # Converts strings for HTTP POST requests on inference
import numpy as np  # For performing matrix operations and numerical processing
import pandas as pd  # For manipulating tabular data
import os

region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')
```

Next Step

Get the Amazon SageMaker Execution Role (p. 561)

Get the Amazon SageMaker Execution Role

Get the execution role for the notebook instance. This is the IAM role that you created when you created your notebook instance. You pass the role to the tuning job.

```python
from sagemaker import get_execution_role

role = get_execution_role()
print(role)
```

Next Step

Specify a Bucket and Data Output Location (p. 561)

Specify a Bucket and Data Output Location

Specify the name of the Amazon S3 bucket where you want to store the output of the training jobs that the tuning job launches. The name of the bucket must contain `sagemaker`, and be globally unique. The bucket must be in the same AWS Region as the notebook instance that you use for this example. You can use the bucket that you created when you set up Amazon SageMaker, or you can create a new bucket. For information, see Step 1: Create an Amazon S3 Bucket (p. 25).

**Note**
The name of the bucket doesn't need to contain `sagemaker` if the role that you use to run the hyperparameter tuning job has a policy that gives the SageMaker service principle `S3FullAccess` permission.
prefix is the path within the bucket where Amazon SageMaker stores the output from training jobs.

```python
bucket = 'sagemaker-MyBucket'  # Replace with the name of your S3 bucket
prefix = 'sagemaker/DEMO-automatic-model-tuning-xgboost-dm'
```

**Next Step**

Download, Prepare, and Upload Training Data (p. 562)

**Download, Prepare, and Upload Training Data**

For this example, you use a training dataset of information about bank customers that includes the customer's job, marital status, and how they were contacted during the bank's direct marketing campaign. To use a dataset for a hyperparameter tuning job, you download it, transform the data, and then upload it to an Amazon S3 bucket.

For more information about the dataset and the data transformation that the example performs, see the `hpo_xgboost_direct_marketing_sagemaker_APIs` notebook in the Hyperparameter Tuning section of the SageMaker Examples tab in your notebook instance.

**Download and Explore the Training Dataset**

To download and explore the dataset, run the following code in your notebook:

```bash
!unzip -o bank-additional.zip
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)  # Make sure we can see all of the columns
data.set_option('display.max_rows', 5)  # Keep the output on one page
data
```

**Prepare and Upload Data**

Before creating the hyperparameter tuning job, prepare the data and upload it to an S3 bucket where the hyperparameter tuning job can access it.

Run the following code in your notebook:

```python
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)  # Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'unemployed']), 1, 0)  # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)  # Convert categorical variables to sets of indicators
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)
train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data)), int(0.9*len(model_data))])
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)]).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)]).to_csv('validation.csv', index=False, header=False)
pd.concat([test_data['y_yes'], test_data.drop(['y_no', 'y_yes'], axis=1)]).to_csv('test.csv', index=False, header=False)
```
Next Step

Configure and Launch a Hyperparameter Tuning Job (p. 563)

**Configure and Launch a Hyperparameter Tuning Job**

To configure and launch a hyperparameter tuning job, complete the following steps.

**Topics**
- Specify the Hyperparameter Tuning Job Settings (p. 563)
- Configure the Training Jobs (p. 564)
- Name and Launch the Hyperparameter Tuning Job (p. 565)
- Next Step (p. 566)

**Specify the Hyperparameter Tuning Job Settings**

To specify settings for the hyperparameter tuning job, you define a JSON object. You pass the object as the value of the `HyperParameterTuningJobConfig` parameter to `CreateHyperParameterTuningJob` (p. 890) when you create the tuning job.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune. For more information, see Define Hyperparameter Ranges (p. 558)
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An **objective metric** is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

**Note**
To use your own algorithm for hyperparameter tuning, you need to define metrics for your algorithm. For information, see Define Metrics (p. 557).

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the XGBoost Algorithm (p. 422) built-in algorithm. The objective metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```python
tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            { "MaxValue": "1", "MinValue": "0", "Name": "eta" },
            { "MaxValue": "2", "MinValue": "0", "Name": "alpha" },
            { "MaxValue": "10", "MinValue": "0", "Name": "min_child_weight" },
            { "MaxValue": "10", "MinValue": "0", "Name": "max_depth" }
        ]
    }
}```
Configure the Training Jobs

To configure the training jobs that the tuning job launches, define a JSON object that you pass as the value of the TrainingJobDefinition parameter of the CreateHyperParameterTuningJob (p. 890) call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.
  
  **Note**
  
  Define metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don't specify metrics. For information about defining metrics, see Define Metrics (p. 557).

- The container image that specifies the training algorithm.
- The input configuration for your training and test data.
- The storage location for the algorithm's output. Specify the S3 bucket where you want to store the output of the training jobs.
- The values of algorithm hyperparameters that are not tuned in the tuning job.
- The type of instance to use for the training jobs.
- The stopping condition for the training jobs. This is the maximum duration for each training job.

In this example, we set static values for the eval_metric, num_round, objective, rate_drop, and tweedie_variance_power parameters of the XGBoost Algorithm (p. 422) built-in algorithm.

```python
from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(boto3.Session().region_name, 'xgboost')
s3_input_train = 's3://{}/{}/train'.format(bucket, prefix)
s3_input_validation = 's3://{}/{}/validation/'.format(bucket, prefix)
```
training_job_definition = {
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train
                }
            }
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation
                }
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://{}//{}//output".format(bucket,prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.2xlarge",
        "VolumeSizeInGB": 10
    },
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 43200
    }
}

**Name and Launch the Hyperparameter Tuning Job**

Now you can provide a name for the hyperparameter tuning job and then launch it by calling the CreateHyperParameterTuningJob (p. 890) API. Pass `tuning_job_config` and `training_job_definition` that you created in previous steps as the values of the parameters.

```python
name = "MyTuningJob"
smlclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = name,
                                           HyperParameterTuningJobConfig = tuning_job_config,
                                           TrainingJobDefinition = training_job_definition)
```
Next Step

Monitor the Progress of a Hyperparameter Tuning Job (p. 566)

Monitor the Progress of a Hyperparameter Tuning Job

To monitor the progress of a hyperparameter tuning job and the training jobs that it launches, use the Amazon SageMaker console.

Topics
- View the Status of the Hyperparameter Tuning Job (p. 566)
- View the Status of the Training Jobs (p. 566)
- View the Best Training Job (p. 567)

View the Status of the Hyperparameter Tuning Job

To view the status of the hyperparameter tuning job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Hyperparameter tuning jobs.
3. In the list of hyperparameter tuning jobs, check the status of the hyperparameter tuning job you launched. A tuning job can be:
   - Completed—The hyperparameter tuning job successfully completed.
   - InProgress—The hyperparameter tuning job is in progress. One or more training jobs are still running.
   - Failed—The hyperparameter tuning job failed.
   - Stopped—The hyperparameter tuning job was manually stopped before it completed. All training jobs that the hyperparameter tuning job launched are stopped.
   - Stopping—The hyperparameter tuning job is in the process of stopping.

View the Status of the Training Jobs

To view the status of the training jobs that the hyperparameter tuning job launched

1. In the list of hyperparameter tuning jobs, choose the job that you launched.
2. Choose Training jobs.
3. View the status of each training job. To see more details about a job, choose it in the list of training jobs. To view a summary of the status of all of the training jobs that the hyperparameter tuning job launched, see Training job status counter.

A training job can be:

- **Completed**—The training job successfully completed.
- **InProgress**—The training job is in progress.
- **Stopped**—The training job was manually stopped before it completed.
- **Failed (Retriable)**—The training job failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
- **Failed (Non-retriable)**—The training job failed and can’t be retried. A failed training job can’t be retried when a client error occurs.

View the Best Training Job

A hyperparameter tuning job uses the objective metric that each training job returns to evaluate training jobs. While the hyperparameter tuning job is in progress, the best training job is the one that has returned the best objective metric so far. After the hyperparameter tuning job is complete, the best training job is the one that returned the best objective metric.

To view the best training job, choose Best training job.
To deploy the best training job as a model that you can host at an Amazon SageMaker endpoint, choose **Create model**.

### Next Step

#### Clean up (p. 568)

**Clean up**

To avoid incurring unnecessary charges, when you are done with the example, use the AWS Management Console to delete the resources that you created for it.

**Note**

If you plan to explore other examples, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at [https://console.aws.amazon.com/sagemaker/](https://console.aws.amazon.com/sagemaker/) and delete the notebook instance. Stop the instance before deleting it.
2. Open the Amazon S3 console at [https://console.aws.amazon.com/s3/](https://console.aws.amazon.com/s3/) and delete the bucket that you created to store model artifacts and the training dataset.
3. Open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/) and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at [https://console.aws.amazon.com/cloudwatch/](https://console.aws.amazon.com/cloudwatch/) and delete all of the log groups that have names starting with `/aws/sagemaker/`.

### Stop Training Jobs Early

Stop the training jobs that a hyperparameter tuning job launches early when they are not improving significantly as measured by the objective metric. Stopping training jobs early can help reduce compute time and helps you avoid overfitting your model. To configure a hyperparameter tuning job to stop training jobs early, do one of the following:
- If you are using the AWS SDK for Python (Boto 3), set the TrainingJobEarlyStoppingType field of the HyperParameterTuningJobConfig (p. 1389) object that you use to configure the tuning job to AUTO.
- If you are using the Amazon SageMaker Python SDK, set the early_stopping_type parameter of the HyperParameterTuner object to Auto.
- In the Amazon SageMaker console, in the Create hyperparameter tuning job workflow, under Early stopping, choose Auto.

For a sample notebook that demonstrates how to use early stopping, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_early_stopping/hpo_image_classification_early_stopping.ipynb or open the hpo_image_classification_early_stopping.ipynb notebook in the Hyperparameter Tuning section of the SageMaker Examples in a notebook instance. For information about using sample notebooks in a notebook instance, see Use Example Notebooks (p. 208).

How Early Stopping Works

When you enable early stopping for a hyperparameter tuning job, Amazon SageMaker evaluates each training job the hyperparameter tuning job launches as follows:

- After each epoch of training, get the value of the objective metric.
- Compute the running average of the objective metric for all previous training jobs up to the same epoch, and then compute the median of all of the running averages.
- If the value of the objective metric for the current training job is worse (higher when minimizing or lower when maximizing the objective metric) than the median value of running averages of the objective metric for previous training jobs up to the same epoch, Amazon SageMaker stops the current training job.

Algorithms That Support Early Stopping

To support early stopping, an algorithm must emit objective metrics for each epoch. The following built-in Amazon SageMaker algorithms support early stopping:

- **Linear Learner Algorithm (p. 328)**—Supported only if you use objective_loss as the objective metric.
- **XGBoost Algorithm (p. 422)**
- **Image Classification Algorithm (p. 271)**
- **Object Detection Algorithm (p. 365)**
- **Sequence-to-Sequence Algorithm (p. 410)**
- **IP Insights Algorithm (p. 297)**

**Note**

This list of built-in algorithms that support early stopping is current as of December 13, 2018. Other built-in algorithms might support early stopping in the future. If an algorithm emits a metric that can be used as an objective metric for a hyperparameter tuning job (preferably a validation metric), then it supports early stopping.

To use early stopping with your own algorithm, you must write your algorithms such that it emits the value of the objective metric after each epoch. The following list shows how you can do that in different frameworks:
Types of Warm Start Tuning Jobs

There are two different types of warm start tuning jobs:

Reasons you might want to consider warm start include:

- You want to gradually increase the number of training jobs over several tuning jobs based on the results you see after each iteration.
- You get new data, and want to tune a model using the new data.
- You want to change the ranges of hyperparameters that you used in a previous tuning job, change static hyperparameters to tunable, or change tunable hyperparameters to static values.
- You stopped a previous hyperparameter job early or it stopped unexpectedly.

Topics

- Types of Warm Start Tuning Jobs (p. 570)
- Warm Start Tuning Restrictions (p. 571)
- Warm Start Tuning Sample Notebook (p. 572)
- Create a Warm Start Tuning Job (p. 572)
IDENTICAL_DATA_AND_ALGORITHM

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed.

Use identical data and algorithm when you use the same training data as you used in a previous hyperparameter tuning job, but you want to increase the total number of training jobs or change ranges or values of hyperparameters.

When you run an warm start tuning job of type IDENTICAL_DATA_AND_ALGORITHM, there is an additional field in the response to DescribeHyperParameterTuningJob (p. 1026) named OverallBestTrainingJob. The value of this field is the TrainingJobSummary (p. 1531) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

TRANSFER_LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The training algorithm image can also be a different version from the version used in the parent hyperparameter tuning job. When you use transfer learning, changes in the dataset or the algorithm that significantly affect the value of the objective metric might reduce the usefulness of using warm start tuning.

Warm Start Tuning Restrictions

The following restrictions apply to all warm start tuning jobs:

- A tuning job can have a maximum of 5 parent jobs, and all parent jobs must be in a terminal state (Completed, Stopped, or Failed) before you start the new tuning job.
- The objective metric used in the new tuning job must be the same as the objective metric used in the parent jobs.
- The total number of static plus tunable hyperparameters must remain the same between parent jobs and the new tuning job. Because of this, if you think you might want to use a hyperparameter as tunable in a future warm start tuning job, you should add it as a static hyperparameter when you create a tuning job.
- The type of each hyperparameter (continuous, integer, categorical) must not change between parent jobs and the new tuning job.
- The number of total changes from tunable hyperparameters in the parent jobs to static hyperparameters in the new tuning job, plus the number of changes in the values of static hyperparameters cannot be more than 10. Each value in a categorical hyperparameter counts against this limit. For example, if the parent job has a tunable categorical hyperparameter with the possible values red and blue, you change that hyperparameter to static in the new tuning job, that counts as 2 changes against the allowed total of 10. If the same hyperparameter had a static value of red in the parent job, and you change the static value to blue in the new tuning job, it also counts as 2 changes.
- Warm start tuning is not recursive. For example, if you create MyTuningJob3 as a warm start tuning job with MyTuningJob2 as a parent job, and MyTuningJob2 is itself a warm start tuning job with a parent job MyTuningJob1, the information that was learned when running MyTuningJob1 is not used for MyTuningJob3. If you want to use the information from MyTuningJob1, you must explicitly add it as a parent for MyTuningJob3.
• The training jobs launched by every parent job in a warm start tuning job count against the 500 maximum training jobs for a tuning job.
• Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Warm Start Tuning Sample Notebook

For a sample notebook that shows how to use warm start tuning, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_warmstart/hpo_image_classification_warmstart.ipynb. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Example Notebooks (p. 208). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The warm start tuning example notebook is located in the Hyperparameter tuning section, and is named hpo_image_classification_warmstart.ipynb. To open a notebook, click on its Use tab and select Create copy.

Create a Warm Start Tuning Job

You can use either the low-level AWS SDK for Python (Boto 3) or the high-level Amazon SageMaker Python SDK to create a warm start tuning job.

Topics
• Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3)) (p. 572)
• Create a Warm Start Tuning Job (Amazon SageMaker Python SDK) (p. 573)

Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3))

To use warm start tuning, you specify the values of a HyperParameterTuningJobWarmStartConfig (p. 1394) object, and pass that as the WarmStartConfig field in a call to CreateHyperParameterTuningJob (p. 890).

The following code shows how to create a HyperParameterTuningJobWarmStartConfig (p. 1394) object and pass it to CreateHyperParameterTuningJob (p. 890) job by using the low-level Amazon SageMaker API for Python (Boto 3).

Create the HyperParameterTuningJobWarmStartConfig object:

```python
warm_start_config = {
    "ParentHyperParameterTuningJobs" : [
        {"HyperParameterTuningJobName" : 'MyParentTuningJob'}
    ],
    "WarmStartType" : "IdenticalDataAndAlgorithm"
}
```

Create the warm start tuning job:

```python
smclient = boto3.Session().client('sagemaker')
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = 'MyWarmStartTuningJob',
                                           HyperParameterTuningJobConfig = tuning_job_config, # See notebook for tuning configuration
                                           TrainingJobDefinition = training_job_definition, # See notebook for job definition
                                           # Other parameters...
)```
Create a Warm Start Tuning Job (Amazon SageMaker Python SDK)

To use the Amazon SageMaker Python SDK to run a warm start tuning job, you:

- Specify the parent jobs and the warm start type by using a `WarmStartConfig` object.
- Pass the `WarmStartConfig` object as the value of the `warm_start_config` argument of a `HyperparameterTuner` object.
- Call the `fit` method of the `HyperparameterTuner` object.

For more information about using the Amazon SageMaker Python SDK for hyperparameter tuning, see [https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning](https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning).

This example uses an estimator that uses the [Image Classification Algorithm](https://docs.aws.amazon.com/sagemaker/latest/dg/image-classification-algorithm.html) algorithm for training. The following code sets the hyperparameter ranges that the warm start tuning job searches within to find the best combination of values. For information about setting hyperparameter ranges, see [Define Hyperparameter Ranges](https://docs.aws.amazon.com/sagemaker/latest/dg/define-hyperparameter-range.html).

```python
hyperparameter_ranges = {'learning_rate': ContinuousParameter(0.0, 0.1),
                        'momentum': ContinuousParameter(0.0, 0.99)}
```

The following code configures the warm start tuning job by creating a `WarmStartConfig` object.

```python
from sagemaker.tuner import WarmStartConfig,
                     WarmStartTypes

parent_tuning_job_name = "MyParentTuningJob"
warm_start_config = WarmStartConfig(type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
                                     parents={parent_tuning_job_name})
```

Now set the values for static hyperparameters, which are hyperparameters that keep the same value for every training job that the warm start tuning job launches. In the following code, `imageclassification` is an estimator that was created previously.

```python
imageclassification.set_hyperparameters(num_layers=18,
                                        num_classes=257,
                                        num_training_samples=15420,
                                        mini_batch_size=128,
                                        epochs=30,
                                        optimizer='sgd',
                                        top_k='2',
                                        precision_dtype='float32',
                                        augmentation_type='crop')
```

Now create the `HyperparameterTuner` object and pass the `WarmStartConfig` object that you previously created as the `warm_start_config` argument.

```python
tuner_warm_start = HyperparameterTuner(imageclassification,
                                        'validation:accuracy',
                                        hyperparameter_ranges,
                                        objective_type='Maximize',
                                        max_jobs=10,
                                        max_parallel_jobs=2,
                                        base_tuning_job_name='warmstart',
                                        warm_start_config=warm_start_config)
```
Finally, call the `fit` method of the `HyperparameterTuner` object to launch the warm start tuning job.

```python
 tuner_warm_start.fit(
    {'train': s3_input_train, 'validation': s3_input_validation},
    include_cls_metadata=False)
```

## Automatic Model Tuning Resource Limits

Amazon SageMaker sets default limits for the following resources:

- Number of concurrent hyperparameter tuning jobs - 100
- Number of hyperparameters that can be searched - 20
- Number of metrics defined per hyperparameter tuning job - 20
- Number of concurrent training jobs per hyperparameter tuning job - 10
- Number of training jobs per hyperparameter tuning job - 500
- Maximum run time for a hyperparameter tuning job - 30 days

When you plan hyperparameter tuning jobs, you also have to take the limits on training resources into account. For information about the default resource limits for Amazon SageMaker training jobs, see Amazon SageMaker Limits. Every concurrent training instance that all of your hyperparameter tuning jobs run on count against the total number of training instances allowed. For example, suppose you run 10 concurrent hyperparameter tuning jobs. Each of those hyperparameter tuning jobs runs 100 total training jobs, and runs 20 concurrent training jobs. Each of those training jobs runs on one `ml.m4.xlarge` instance. The following limits apply:

- Number of concurrent hyperparameter tuning jobs - You don't need to increase the limit, because 10 tuning jobs is below the limit of 100.
- Number of training jobs per hyperparameter tuning job - You don't need to increase the limit, because 100 training jobs is below the limit of 500.
- Number of concurrent training jobs per hyperparameter tuning job - You need to request a limit increase to 20, because the default limit is 10.
- Amazon SageMaker training `ml.m4.xlarge` instances - You need to request limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.
- Amazon SageMaker training total instance count - You need to request a limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.

For information about requesting limit increases for AWS resources, see AWS Service Limits.

## Best Practices for Hyperparameter Tuning

Hyperparameter optimization is not a fully-automated process. To improve optimization, use the following guidelines when you create hyperparameters.

**Topics**

- Choosing the Number of Hyperparameters (p. 575)
Choosing the Number of Hyperparameters

The difficulty of a hyperparameter tuning job depends primarily on the number of hyperparameters that Amazon SageMaker has to search. Although you can simultaneously use up to 20 variables in a hyperparameter tuning job, limiting your search to a much smaller number is likely to give better results.

Choosing Hyperparameter Ranges

The range of values for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. Although you might want to specify a very large range that covers every possible value for a hyperparameter, you will get better results by limiting your search to a small range of values. If you get the best metric values within a part of a range, consider limiting the range to that part.

Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, Amazon SageMaker attempts to figure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they should be log-scaled, it might take some time for Amazon SageMaker to discover that. If you know that a hyperparameter should be log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

Choosing the Best Number of Concurrent Training Jobs

Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.

Running Training Jobs on Multiple Instances

When a training job runs on multiple instances, hyperparameter tuning uses the last-reported objective metric from all instances of that training job as the value of the objective metric for that training job. Design distributed training jobs so that you get they report the objective metric that you want.

Tune Multiple Algorithms to Find the Best Model

When you create a new hyperparameter optimization (HPO) job with Amazon SageMaker, you have the option of using the console or the API. You provide one or more job specifications for the different algorithms you’re testing. These are called training definitions. Each training definition has a name, an algorithm source, metrics selection, an objective metric, and a configuration for a set of hyperparameter values. It also has a data configuration for setting up the input data channels for the algorithm you choose, and a setting for the output data location. You select the resources you want to use for the training run.

Topics
- Get Started (p. 576)
- Managing Hyperparameter Tuning Jobs (p. 576)
• Create a new single or multi-algorithm HPO tuning job (p. 576)

Get Started

Using Multi-Algorithm HPO

To use multi-algorithm HPO you must add more than one training definition to your hyperparameter tuning job. Each training definition holds the configuration options for each algorithm you want to try.

In the console, you add training definitions when you create the HPO tuning job by choosing Add training definition, and then following the configuration steps for each algorithm that you want to use.

When you start the configuration steps, please note that the warm start and early stopping features are not available with multi-algorithm HPO. If you want to use these features, you can only tune a single algorithm at a time.

If you’re using an API request, instead of the single TrainingJobDefinition, you must provide a list of training definitions using TrainingJobDefinitions. You must use one or the other, not both.

Managing Hyperparameter Tuning Jobs

You can clone a job, add or edit tags, or create a new hyperparameter tuning job from the console. You can also use the search feature to find jobs by their name, creation time, and status.

Creating a Hyperparameter Tuning Job

To create a new job, open the Amazon SageMaker console, choose Training, choose Hyperparameter tuning jobs, and then choose Create hyperparameter tuning job.

For instructions on using the API to create a tuning job, see Example: Hyperparameter Tuning Job.

Cloning an existing training Job

You can save time by cloning a training job, which copies all of the job’s settings, including data channels, S3 bucket locations, algorithms, and the hyperparameter options.

To clone a training job

• On the Training jobs page or on the Hyperparameter tuning jobs page. choose Actions and then choose Clone.

Editing Tags

You enter tags as key-value pairs. Values are not required. You can use just the key. To see the keys associated with a job, choose the Tags tab on the tuning job’s details page.

Create a new single or multi-algorithm HPO tuning job

Defining job settings

Your tuning job settings are applied across all of the algorithms in the HPO tuning job. Warm start and early stopping are available only when tuning a single algorithm. After you define the job settings you will create individual training definitions for each algorithm or variation you want to tune.
Warm Start

If you cloned this job, you can choose to use the results from a previous tuning job to improve the performance of this tuning job. This is the warm start feature and it is only available when tuning a single algorithm. When you choose this option, you can choose up to five previous hyperparameter tuning jobs to use. Alternatively, you can use transfer learning to add additional data to the parent tuning job. When you select this option, you choose one previous tuning job as the parent.

Warm start is compatible with tuning jobs created after October 1, 2018. For more information, see Run a warm start job.

Early Stopping

Early stopping stops training jobs when they are unlikely to improve the current best objective metric of the hyperparameter tuning job. Like warm start, this feature is only available when tuning a single algorithm. This is an automatic feature without configuration options, and it's disabled by default.

Tuning Strategy

Tuning strategy can be either random or bayesian. It specifies how the automatic tuning searches over specified hyperparameter ranges. You specify the ranges in a later step. For more information, see How Hyperparameter Tuning Works.

Training Definitions

You must provide at least one training definition for each training job. Each training definition specifies the configuration for an algorithm. To create several definitions for your training job you can clone a definition.

Name

Provide a unique name for the training definition.

Permissions

Amazon SageMaker requires permissions to call other services on your behalf. Choose an IAM role or let AWS create a role that has the AmazonSageMakerFullAccess IAM policy attached.

Optional Security Settings

The network isolation setting prevents the container from making any outbound network calls. This is required for AWS Marketplace machine learning offerings.

You can also choose to use a private VPC.

Note

Inter-container encryption is only available when creating job definitions from the API.

Algorithm Options

You can choose one of the built-in algorithms, your own algorithm, your own container with an algorithm, or you can subscribe to an algorithm from AWS Marketplace.

If you choose a built-in algorithm, it has the ECR image information prepopulated. If you choose your own container, you must specify the ECR image information. You can select the input mode for the algorithm as file or pipe. If you plan to supply your data using a .CSV file from Amazon S3, you should select the file.

Metrics
Use Reinforcement Learning with Amazon SageMaker

Reinforcement learning (RL) is a machine learning technique that attempts to learn a strategy, called a policy, that optimizes an objective for an agent acting in an environment. For example, the agent might be a robot, the environment might be a maze, and the goal might be to successfully navigate the maze in the smallest amount of time. In RL, the agent takes an action, observes the state of the...
Why is Reinforcement Learning Important?

RL is well-suited for solving large, complex problems. For example, supply chain management, HVAC systems, industrial robotics, game artificial intelligence, dialog systems, and autonomous vehicles. Because RL models learn by a continuous process of receiving rewards and punishments for every action taken by the agent, it is possible to train systems to make decisions under uncertainty and in dynamic environments.

Markov Decision Process (MDP)

RL is based on models called Markov Decision Processes (MDPs). An MDP consists of a series of time steps. Each time step consists of the following:

Environment

Defines the space in which the RL model operates. This can be either a real-world environment or a simulator. For example, if you train a physical autonomous vehicle on a physical road, that would be a real-world environment. If you train a computer program that models an autonomous vehicle driving on a road, that would be a simulator.

State

Specifies all information about the environment and past steps that is relevant to the future. For example, in an RL model in which a robot can move in any direction at any time step, then the position of the robot at the current time step is the state, because if we know where the robot is, it isn't necessary to know the steps it took to get there.

Action

What the agent does. For example, the robot takes a step forward.

Reward

A number that represents the value of the state that resulted from the last action that the agent took. For example, if the goal is for a robot to find treasure, the reward for finding treasure might be 5, and the reward for not finding treasure might be 0. The RL model attempts to find a strategy that optimizes the cumulative reward over the long term. This strategy is called a policy.

Observation

Information about the state of the environment that is available to the agent at each step. This might be the entire state, or it might be just a part of the state. For example, the agent in a chess-
playing model would be able to observe the entire state of the board at any step, but a robot in a maze might only be able to observe a small portion of the maze that it currently occupies.

Typically, training in RL consists of many episodes. An episode consists of all of the time steps in an MDP from the initial state until the environment reaches the terminal state.

**Key Features of Amazon SageMaker RL**

To train RL models in Amazon SageMaker RL, use the following components:

- An RL environment. You can use custom environments, open-source environments, or commercial environments. For information, see [RL Environments in Amazon SageMaker](p. 583).

The following diagram shows the RL components that are supported in Amazon SageMaker RL.
Key Features of Amazon SageMaker RL

- End-to-end examples for classic RL and real-world problems
- AWS Simulation Environments:
  - Amazon Sumerian
  - AWS RoboMaker
- Open Source Environments:
  - EnergyPlus
  - RoboSchool
  - PyBullet
- Open AI Gym
- RL Toolkits that provide RL agent algorithm implementations:
  - RL-Coach:
    - DQN
    - PPO
    - HER
    - Rainbow
  - RL-Ray RLLib:
    - APEX
    - ES
    - IMPALA
- SageMaker Deep Learning Frameworks:
  - TensorFlow
  - MxNet
  - PyTorch
- Training Options:
  - Single Machine / Distributed
  - Local / Remote simulation

SageMaker supported

Customer BYO
Sample RL Workflow Using Amazon SageMaker RL

The following example describes the steps for developing RL models using Amazon SageMaker RL.

For complete code examples, see the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning.

1. **Formulate the RL problem**—First, formulate the business problem into an RL problem. For example, auto scaling enables services to dynamically increase or decrease capacity depending on conditions that you define. Currently, this requires setting up alarms, scaling policies, and thresholds, and other manual steps. To solve this with RL, we define the components of the Markov Decision Process:

   a. **Objective**—Scale instance capacity so that it matches the desired load profile.
   
   b. **Environment**—A custom environment that includes the load profile. It generates a simulated load with daily and weekly variations and occasional spikes. The simulated system has a delay between when new resources are requested and when they become available for serving requests.
   
   c. **State**—The current load, number of failed jobs, and number of active machines
   
   d. **Action**—Remove, add, or keep the same number of instances.
   
   e. **Reward**—A positive reward for successful transactions, a high penalty for failing transactions beyond a specified threshold.

2. **Define the RL environment**—The RL environment can be the real world where the RL agent interacts or a simulation of the real world. You can connect open source and custom environments developed using Gym interfaces, and commercial simulation environments such as MATLAB and Simulink.

3. **Define the presets**—The presets configure the RL training jobs and define the hyperparameters for the RL algorithms.

4. **Write the training code**—Write training code as a Python script and pass the script to an Amazon SageMaker training job. In your training code, import the environment files and the preset files, and then define the \texttt{main()} function.

5. **Train the RL Model**—Use the Amazon SageMaker \texttt{RLEstimator} in the Amazon SageMaker Python SDK to start an RL training job. If you are using local mode, the training job runs on the notebook instance. When you use Amazon SageMaker for training, you can select GPU or CPU instances. Store the output from the training job in a local directory if you train in local mode, or on Amazon S3 if you use Amazon SageMaker training.

   For information about using the Amazon SageMaker Python SDK for RL, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/rl/README.rst .

   The \texttt{RLEstimator} requires the following information as parameters:

   a. The source directory where the environment, presets, and training code are uploaded.
   
   b. The path to the training script.
   
   c. The RL toolkit and deep learning framework you want to use. This automatically resolves to the Amazon ECR path for the RL container.
   
   d. The training parameters, such as the instance count, job name, and S3 path for output.
   
   e. Metric definitions that you want to capture in your logs. These can also be visualized in CloudWatch and in Amazon SageMaker notebooks.

6. **Visualize training metrics and output**—After a training job that uses an RL model completes, you can view the metrics you defined in the training jobs in CloudWatch. You can also plot the metrics in a notebook by using the Amazon SageMaker Python SDK analytics library. Visualizing metrics helps you understand how the performance of the model as measured by the reward improves over time.

   \textbf{Note}
   
   If you train in local mode, you can't visualize metrics in CloudWatch.
7. **Evaluate the model**—Check pointed data from the previously trained models can be passed on for evaluation and inference in the checkpoint channel. In local mode, use the local directory. In Amazon SageMaker training mode, you need to upload the data to S3 first.

8. **Deploy RL models**—Finally, deploy the trained model on an endpoint hosted on Amazon SageMaker or on an Edge device by using AWS IoT Greengrass.

## RL Environments in Amazon SageMaker

Amazon SageMaker RL uses environments to mimic real-world scenarios. Given the current state of the environment and an action taken by the agent or agents, the simulator processes the impact of the action, and returns the next state and a reward. Simulators are useful in cases where it is not safe to train an agent in the real world (for example, flying a drone) or if the RL algorithm takes a long time to converge (for example, when playing chess).

The following diagram shows an example of the interactions with a simulator for a car racing game.

![Diagram of RL Environments in Amazon SageMaker](image)

The simulation environment consists of an agent and a simulator. Here, a convolutional neural network (CNN) consumes images from the simulator and generates actions to control the game controller. With multiple simulations, this environment generates training data of the form `state_t, action, state_{t+1}`, and `reward_{t+1}`. Defining the reward is not trivial and impacts the RL model quality. We want to provide a few examples of reward functions, but would like to make it user-configurable.

### Topics

- Use OpenAI Gym Interface for Environments in Amazon SageMaker RL (p. 584)
- Use Open Source Environments (p. 584)
- Use Commercial Environments (p. 584)
Use OpenAI Gym Interface for Environments in Amazon SageMaker RL

To use OpenAI Gym environments in Amazon SageMaker RL, use the following API elements. For more information about OpenAI Gym, see https://gym.openai.com/docs/.

- **env.action_space**—Defines the actions the agent can take, specifies whether each action is continuous or discrete, and specifies the minimum and maximum if the action is continuous.
- **env.observation_space**—Defines the observations the agent receives from the environment, as well as minimum and maximum for continuous observations.
- **env.reset()**—Initializes a training episode. The reset() function returns the initial state of the environment, and the agent uses the initial state to take its first action. The action is then sent to the step() repeatedly until the episode reaches a terminal state. When step() returns `done = True`, the episode ends. The RL toolkit re-initializes the environment by calling reset().
- **step()**—Takes the agent action as input and outputs the next state of the environment, the reward, whether the episode has terminated, and an info dictionary to communicate debugging information. It is the responsibility of the environment to validate the inputs.
- **env.render()** Used for environments that have visualization. The RL toolkit calls this function to capture visualizations of the environment after each call to the step() function.

Use Open Source Environments

You can use open source environments, such as EnergyPlus and RoboSchool, in Amazon SageMaker RL by building your own container. For more information about EnergyPlus, see https://energyplus.net/. For more information about RoboSchool, see https://github.com/openai/roboschool. The HVAC and RoboSchool examples in the samples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning show how to build a custom container to use with Amazon SageMaker RL:

Use Commercial Environments

You can use commercial environments, such as MATLAB and Simulink, in Amazon SageMaker RL by building your own container. You need to manage your own licenses.

Distributed Training with Amazon SageMaker RL

Amazon SageMaker RL supports multi-core and multi-instance distributed training. Depending on your use case, training and/or environment rollout can be distributed. For example, Amazon SageMaker RL works for the following distributed scenarios:

- Single training instance and multiple rollout instances of the same instance type. For an example, see the Neural Network Compression example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance and multiple rollout instances, where different instance types for training and rollouts. For an example, see the AWS DeepRacer / AWS RoboMaker example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance that uses multiple cores for rollout. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning. This is useful if the simulation environment is light-weight and can run on a single thread.
Hyperparameter Tuning with Amazon SageMaker RL

You can run a hyperparameter tuning job to optimize hyperparameters for Amazon SageMaker RL. The Roboschool example in the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning shows how you can do this with RL Coach. The launcher script shows how you can abstract parameters from the Coach preset file and optimize them.

Train a Deep Graph Network

In this overview, you learn how to get started with a deep graph network by using one of the DGL containers in Amazon Elastic Container Registry (Amazon ECR). You can also see links to practical examples for deep graph networks.

What Is a Deep Graph Network?

Deep graph networks refer to a type of neural network that is trained to solve graph problems. A deep graph network uses an underlying deep learning framework like PyTorch or MXNet. The potential for graph networks in practical AI applications are highlighted in the Amazon SageMaker tutorials for Deep Graph Library (DGL). Examples for training models on graph datasets include social networks, knowledge bases, biology, and chemistry.
Several examples are provided using Amazon SageMaker’s deep learning containers that are preconfigured with DGL. If you have special modules you want to use with DGL, you can also build your own container. The examples involve heterographs, which are graphs that have multiple types of nodes and edges, and draw on a variety of applications across disparate scientific fields, such as bioinformatics and social network analysis. DGL provides a wide array of graph neural network implementations for different types models. Some of the highlights include:

- GCN - Graph convolutional network
- R-GCN - Relational graph convolutional network
- GAT - Graph attention network
- DGMG - Deep generative models of graphs
- JTNN - Junction tree neural network

Get Started

DGL is available as a deep learning container in Amazon ECR. You can select deep learning containers when you write your estimator function in an Amazon SageMaker notebook. You can also craft your own custom container with DGL by following the Bring Your Own Container guide. The easiest way to get started with a deep graph network uses one of the DGL containers in Amazon ECR.

Note
Backend framework support is limited to PyTorch and MXNet.

Setup

If you are using Amazon SageMaker Studio, you need to clone the examples repo first. If you are using a notebook instance, you can find the examples choosing the SageMaker icon at bottom of the left toolbar.

To clone the Amazon SageMaker SDK and notebook examples repository

1. From the Jupyter Lab view in Amazon SageMaker, go to the File Browser at the top of the left toolbar. From the file browser panel you can see a new navigation at the top of the panel.
2. Choose the icon on the far right to clone a git repository.
3. Add the repository URL: https://github.com/awslabs/amazon-sagemaker-examples.git
4. Browse the newly added folder and its contents. The DGL examples are stored in the sagemaker-python-sdk folder.

Run a Graph Network Training Example

To train a deep graph network

1. From the Jupyter Lab view in Amazon SageMaker, browse the example notebooks and look for dgl folders. Several files may be included to support an example. Examine the README for any prerequisites.
2. Run the .ipynb notebook example.
3. Find the estimator function, and note the line where it is using an Amazon ECR container for DGL and a specific instance type. You may want to update this to use a container in your preferred Region.
4. Run the function to launch the instance and use the DGL container for training a graph network. Charges are incurred for launching this instance. The instance self-terminates when the training is complete.
Examples

An example of knowledge graph embedding (KGE) is provided. It uses the Freebase dataset, a knowledge base of general facts. An example use case would be to graph the relationships of persons and predict their nationality.

An example implementation of a graph convolutional network (GCN) shows how you can train a graph network to predict toxicity. A physiology dataset, Tox21, provides toxicity measurements for how substances affect biological responses.

Another GCN example shows how to train a graph network on a scientific publications bibliography dataset, known as Cora. You can use it to find relationships between authors, topics, and conferences.

The last example is a recommender system for movie reviews. It uses a graph convolutional matrix completion (GCMC) network trained on the MovieLens datasets. These datasets consist of movie titles, genres, and ratings by users.

Use a Deep Learning Container with DGL

The following examples use preconfigured deep learning containers. These are the easiest to try since they work out-of-the-box on Amazon SageMaker.

- Semi-supervised classification of a knowledge base using a GCN
- Learning embeddings of large-scale knowledge graphs using a dataset of scientific publications

Bring Your Own Container with DGL

The following examples enable you to bring your own container (BYOC). Read the BYOC guide and familiarize yourself with that process before trying these. Configuration is required.

- Molecular property prediction of toxicity using a GCN
- Recommender system for movies using a GCMC implementation

Monitor and Analyze Training Jobs Using Metrics

An Amazon SageMaker training job is an iterative process that teaches a model to make predictions by presenting examples from a training dataset. Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help diagnose whether the model is learning well and will generalize well for making predictions on unseen data. The training algorithm writes the values of these metrics to logs, which Amazon SageMaker monitors and sends to Amazon CloudWatch in real time. To analyze the performance of your training job, you can view graphs of these metrics in CloudWatch. When a training job has completed, you can also get a list of the metric values that it computes in its final iteration by calling the DescribeTrainingJob (p. 1066) operation.

Topics

- Training Metrics Sample Notebooks (p. 588)
- Defining Training Metrics (p. 588)
- Monitoring Training Job Metrics (Console) (p. 590)
- Monitoring Training Job Metrics (Amazon SageMaker Console) (p. 590)
- Example: Viewing a Training and Validation Curve (p. 593)
Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics:

- An Introduction to the Amazon SageMaker ObjectToVec Model for Sequence-to-sequence Embedding (object2vec_sentence_similarity.ipynb)
- Regression with the Amazon SageMaker XGBoost Algorithm (xgboost_abalone.ipynb)

For instructions how to create and access Jupyter notebook instances that you can use to run the examples in Amazon SageMaker, see Use Example Notebooks (p. 208). To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the SageMaker Examples tab. To access the example notebooks that show how to use training metrics, object2vec_sentence_similarity.ipynb and xgboost_abalone.ipynb, from the Introduction to Amazon algorithms section. To open a notebook, choose its Use tab, then choose Create copy.

Defining Training Metrics

Amazon SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you want Amazon SageMaker to parse logs from a custom algorithm and send metrics that the algorithm emits to CloudWatch, you have to specify the metrics that you want Amazon SageMaker to send to CloudWatch when you configure the training job. You specify the name of the metrics that you want to send and the regular expressions that Amazon SageMaker uses to parse the logs that your algorithm emits to find those metrics.

You can specify the metrics that you want to track with the Amazon SageMaker console, the Amazon SageMaker Python SDK (https://github.com/aws/sagemaker-python-sdk), or the low-level Amazon SageMaker API.

Topics
- Defining Regular Expressions for Metrics (p. 588)
- Defining Training Metrics (Low-level Amazon SageMaker API) (p. 589)
- Defining Training Metrics (Amazon SageMaker Python SDK) (p. 589)
- Define Training Metrics (Console) (p. 590)

Defining Regular Expressions for Metrics

To find a metric, Amazon SageMaker searches the logs that your algorithm emits and finds logs that match the regular expression that you specify for that metric. If you are using your own algorithm, do the following:

- Make sure that the algorithm writes the metrics that you want to capture to logs
- Define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

For example, suppose your algorithm emits metrics for training error and validation error by writing logs similar to the following to stdout or stderr:

```
Train_error=0.138318; Valid_error = 0.324557;
```

If you want to monitor both of those metrics in CloudWatch, your AlgorithmSpecification would look like the following:
"AlgorithmSpecification": {
  "TrainingImage": ContainerName,
  "TrainingInputMode": "File",
  "MetricDefinitions": [
    { "Name": "train:error",
      "Regex": "Train_error=(.*?);"
    },
    { "Name": "validation:error",
      "Regex": "Valid_error=(.*?);"
    }
  ]
}

In the regex for the train:error metric defined above, the first part of the regex finds the exact text "Train_error=", and the expression (.*?); captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, * means any character, ? means zero or more, and ? means capture only until the first instance of the ; character.

**Defining Training Metrics (Low-level Amazon SageMaker API)**

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions in the `MetricDefinitions` field of the `AlgorithmSpecification` input parameter that you pass to the `CreateTrainingJob` operation. For example, if you want to monitor both the train:error and validation:error metrics in CloudWatch, your AlgorithmSpecification would look like the following:

```
"AlgorithmSpecification": {
  "TrainingImage": ContainerName,
  "TrainingInputMode": "File",
  "MetricDefinitions": [
    { "Name": "train:error",
      "Regex": "Train_error=(.*?);"
    },
    { "Name": "validation:error",
      "Regex": "Valid_error=(.*?);"
    }
  ]
}
```

For more information about defining and running a training job by using the low-level Amazon SageMaker API, see [Create and Run a Training Job (AWS SDK for Python (Boto 3))](p. 32).

**Defining Training Metrics (Amazon SageMaker Python SDK)**

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions as the `metric_definitions` argument when you initialize an `Estimator` object. For example, if you want to monitor both the train:error and validation:error metrics in CloudWatch, your `Estimator` initialization would look like the following:

```
estimator = Estimator(image_name=ImageName,
  role='SageMakerRole', train_instance_count=1,
  train_instance_type='ml.c4.xlarge',
  train_instance_type='ml.c4.xlarge',
```


For more information about training by using Amazon SageMaker Python SDK estimators, see https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

Define Training Metrics (Console)

You can define metrics for a custom algorithm in the console when you create a training job by providing the name and regular expression (regex) for Metrics.

For example, if you want to monitor both the train:error and validation:error metrics in CloudWatch, your metric definitions would look like the following:

```python
metric_definitions=[
    {'Name': 'train:error', 'Regex': 'Train_error=(.*?);'},
    {'Name': 'validation:error', 'Regex': 'Valid_error=(.*?);'}
]
```

Monitoring Training Job Metrics (Console)

You can monitor the metrics that a training job emits in real time in the CloudWatch console.

To monitor training job metrics (CloudWatch console)

2. Choose Metrics, then choose /aws/sagemaker/TrainingJobs.
3. Choose TrainingJobName.
4. On the All metrics tab, choose the names of the training metrics that you want to monitor.
5. On the Graphed metrics tab, configure the graph options. For more information about using CloudWatch graphs, see Graph Metrics in the Amazon CloudWatch User Guide.

Monitoring Training Job Metrics (Amazon SageMaker Console)

You can monitor the metrics that a training job emits in real time by using the Amazon SageMaker console.

To monitor training job metrics (Amazon SageMaker console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Training jobs**, then choose the training job whose metrics you want to see.
3. Choose **TrainingJobName**.
4. In the **Monitor** section, you can review the graphs of instance utilization and algorithm metrics.
Example: Viewing a Training and Validation Curve

Typically, you split the data that you train your model on into training and validation datasets. You use the training set to train the model parameters that are used to make predictions on the training dataset. Then you test how well the model makes predictions by calculating predictions for the validation set. To analyze the performance of a training job, you commonly plot a training curve against a validation curve.

Viewing a graph that shows the accuracy for both the training and validation sets over time can help you to improve the performance of your model. For example, if training accuracy continues to increase over time, but, at some point, validation accuracy starts to decrease, you are likely overfitting your model. To address this, you can make adjustments to your model, such as increasing regularization.

For this example, you can use the Image-classification-full-training example that is in the Example notebooks section of your Amazon SageMaker notebook instance. If you don’t have an Amazon SageMaker notebook instance, create one by following the instructions at Step 2: Create an Amazon SageMaker Notebook Instance (p. 26). If you prefer, you can follow along with the End-to-End Multiclass Image Classification Example in the example notebook on GitHub. You also need an Amazon S3 bucket to store the training data and for the model output. If you haven’t created a bucket to use with Amazon SageMaker, create one by following the instructions at Step 1: Create an Amazon S3 Bucket (p. 25).

To view training and validation error curves

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebooks, and then choose Notebook instances.
3. Choose the notebook instance that you want to use, and then choose Open.
4. On the dashboard for your notebook instance, choose SageMaker Examples.
5. Expand the Introduction to Amazon Algorithms section, and then choose Use next to Image-classification-full-training.ipynb.
7. In the first code cell of the notebook, replace <<bucket-name>> with the name of your S3 bucket.
8. Run all of the cells in the notebook up to the Deploy section. You don’t need to deploy an endpoint or get inference for this example.
10. Choose Metrics, then choose /aws/sagemaker/TrainingJobs.
11. Choose TrainingJobName.
12. On the All metrics tab, choose the train:accuracy and validation:accuracy metrics for the training job that you created in the notebook.
13. On the graph, choose an area that the metric’s values to zoom in. You should see something like the following:
Incremental Training in Amazon SageMaker

Over time, you might find that a model generates inference that are not as good as they were in the past. With incremental training, you can use the artifacts from an existing model and use an expanded dataset to train a new model. Incremental training saves both time and resources.

Use incremental training to:

- Train a new model using an expanded dataset that contains an underlying pattern that was not accounted for in the previous training and which resulted in poor model performance.
- Use the model artifacts or a portion of the model artifacts from a popular publicly available model in a training job. You don't need to train a new model from scratch.
Perform Incremental Training (Console)

To complete this procedure, you need:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 222).
- The URL of the S3 bucket where you've stored the model artifacts that you want to use in incremental training. To find the URL for the model artifacts, see the details page of the training job used to create the model. To find the details page, in the Amazon SageMaker console, choose Inference, choose Models, and then choose the model.

To restart a stopped training job, use the URL to the model artifacts that are stored in the details page as you would with a model or a completed training job.

To perform incremental training (console)

2. In the navigation pane, choose Training, then choose Training jobs.
3. Choose Create training job.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. The training job name must have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 220).
6. (Optional) For Resource configuration, either leave the default values or increase the resource consumption to reduce computation time.
   a. (Optional) For Instance type, choose the ML compute instance type that you want to use. In most cases, ml.m4.xlarge is sufficient.
   b. For Instance count, use the default, 1.
Perform Incremental Training (Console)

7. Provide information about the input data for the training dataset.
   a. For **Channel name**, either leave the default (**train**) or enter a more meaningful name for the training dataset, such as **expanded-training-dataset**.
   b. For **InputMode**, choose **File**. For incremental training, you need to use file input mode.
   c. For **S3 data distribution type**, choose **FullyReplicated**. This causes each ML compute instance to use a full replicate of the expanded dataset when training incrementally.
   d. If the expanded dataset is uncompressed, set the **Compression type** to **None**. If the expanded dataset is compressed using Gzip, set it to **Gzip**.
   e. (Optional) If you are using File input mode, leave **Content type** empty. For Pipe input mode, specify the appropriate MIME type. **Content type** is the multipurpose internet mail extension (MIME) type of the data.
   f. For **Record wrapper**, if the dataset is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO formatted file, choose **None**.
   g. For **S3 data type**, if the dataset is stored as a single file, choose **S3Prefix**. If the dataset is stored as several files in a folder, choose **Manifest**.
   h. For **S3 location**, provide the URL to the path where you stored the expanded dataset.
   i. Choose **Done**.

8. To use model artifacts in a training job, you need to add a new channel and provide the needed information about the model artifacts.
   a. For **Input data configuration**, choose **Add channel**.
   b. For **Channel name**, enter **model** to identify this channel as the source of the model artifacts.
   c. For **InputMode**, choose **File**. Model artifacts are stored as files.
   d. For **S3 data distribution type**, choose **FullyReplicated**. This indicates that each ML compute instance should use all of the model artifacts for training.
   e. For **Compression type**, choose **None** because we are using a model for the channel.
   f. Leave **Content type** empty. Content type is the multipurpose internet mail extension (MIME) type of the data. For model artifacts, we leave it empty.
   g. Set **Record wrapper** to **None** because model artifacts are not stored in RecordIO format.
   h. For **S3 data type**, if you are using a built-in algorithm or an algorithm that stores the model as a single file, choose **S3Prefix**. If you are using an algorithm that stores the model as several files, choose **Manifest**.
   i. For **S3 location**, provide the URL to the path where you stored the model artifacts. Typically, the model is stored with the name **model.tar.gz**. To find the URL for the model artifacts, in the navigation pane, choose **Inference**, then choose **Models**. From the list of models, choose a model to display its details page. The URL for the model artifacts is listed under **Primary container**.
   j. Choose **Done**.

9. For **Output data configuration**, provide the following information:
   a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
   b. (Optional) For **Encryption key**, you can add your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

10. (Optional) For **Tags**, add one or more tags to the training job. A tag is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with...
11. Choose Create training job. Amazon SageMaker creates and runs training job.

After the training job has completed, the newly trained model artifacts are stored under the S3 output path that you provided in the Output data configuration field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 34).

Perform Incremental Training (API)

This example shows how to use Amazon SageMaker APIs to train a model using the Amazon SageMaker image classification algorithm and the Caltech 256 Image Dataset, then train a new model using the first one. It uses Amazon S3 for input and output sources. Please see the incremental training sample notebook for more details on using incremental training.

Note
In this example we used the original datasets in the incremental training, however you can use different datasets, such as ones that contain newly added samples. Upload the new datasets to S3 and make adjustments to the data_channels variable used to train the new model.

Get an AWS Identity and Access Management (IAM) role that grants required permissions and initialize environment variables:

```python
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)
sess = sagemaker.Session()

bucket = sess.default_bucket()
print(bucket)
prefix = 'ic-inr-training'
```

Get the training image for the image classification algorithm:

```python
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sess.boto_region_name, 'image-classification', repo_version="latest")

# Display the training image
print (training_image)
```

Download the training and validation datasets, then upload them to Amazon Simple Storage Service (Amazon S3):

```python
import os
import urllib.request
import boto3

# Define a download function
def download(url):
    filename = url.split("/")[1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)
```
# Download the caltech-256 training and validation datasets
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')

# Create four channels: train, validation, train_lst, and validation_lst
s3train = 's3://{}//{}//train/'.format(bucket, prefix)
s3validation = 's3://{}//{}//validation/'.format(bucket, prefix)

# Upload the first files to the train and validation channels
!aws s3 cp caltech-256-60-train.rec $s3train --quiet
!aws s3 cp caltech-256-60-val.rec $s3validation --quiet

Define the training hyperparameters:

```python
# Define hyperparameters for the estimator
hyperparams = {
    "num_layers": "18",
    "resize": "32",
    "num_training_samples": "50000",
    "num_classes": "10",
    "image_shape": "3,28,28",
    "mini_batch_size": "128",
    "epochs": "3",
    "learning_rate": "0.1",
    "lr_scheduler_step": "2,3",
    "lr_scheduler_factor": "0.1",
    "augmentation_type": "crop_color",
    "optimizer": "sgd",
    "momentum": "0.9",
    "weight_decay": "0.0001",
    "beta_1": "0.9",
    "beta_2": "0.999",
    "gamma": "0.9",
    "eps": "1e-8",
    "top_k": "5",
    "checkpoint_frequency": "1",
    "use_pretrained_model": "0",
    "model_prefix": ""
}
```

Create an estimator object and train the first model using the training and validation datasets:

```python
# Fit the base estimator
s3_output_location = 's3://{}//{}//output/'.format(bucket, prefix)
ic = sagemaker.estimator.Estimator(training_image,
    role,
    train_instance_count=1,
    train_instance_type='ml.p2.xlarge',
    train_volume_size=50,
    train_max_run=360000,
    input_mode='File',
    output_path=s3_output_location,
    sagemaker_session=sess,
    hyperparameters=hyperparams)

train_data = sagemaker.session.s3_input(s3train, distribution='FullyReplicated',
    content_type='application/x-recordio',
    s3_data_type='S3Prefix')
validation_data = sagemaker.session.s3_input(s3validation, distribution='FullyReplicated',
    content_type='application/x-recordio',
    s3_data_type='S3Prefix')
data_channels = {'train': train_data, 'validation': validation_data}
ic.fit(inputs=data_channels, logs=True)
```
To use the model to incrementally train another model, create a new estimator object and use the model artifacts (`ic.model_data`, in this example) for the `model_uri` input argument:

```python
# Given the base estimator, create a new one for incremental training
incr_ic = sagemaker.estimator.Estimator(training_image,
    role,
    train_instance_count=1,
    train_instance_type='ml.p2.xlarge',
    train_volume_size=50,
    train_max_run=360000,
    input_mode='File',
    output_path=s3_output_location,
    sagemaker_session=sess,
    hyperparameters=hyperparams,
    model_uri=ic.model_data) # This parameter will ingest the previous job's model as a new channel
incr_ic.fit(inputs=data_channels, logs=True)
```

After the training job has completed, the newly trained model artifacts are stored under the S3 output path that you provided in `Output_path`. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 34).

---

**Managed Spot Training in Amazon SageMaker**

Amazon SageMaker makes it easy to train machine learning models using managed Amazon EC2 Spot instances. Managed spot training can optimize the cost of training models up to 90% over on-demand instances. Amazon SageMaker manages the Spot interruptions on your behalf.

Managed Spot Training uses Amazon EC2 Spot instance to run training jobs instead of on-demand instances. You can specify which training jobs use spot instances and a stopping condition that specifies how long Amazon SageMaker waits for a job to run using Amazon EC2 Spot instances. Metrics and logs generated during training runs are available in CloudWatch.

Spot instances can be interrupted, causing jobs to take longer to start or finish. You can configure your managed spot training job to use checkpoints. Amazon SageMaker copies checkpoint data from a local path to Amazon S3. When the job is restarted, Amazon SageMaker copies the data from Amazon S3 back into the local path. The training can then resume from the last checkpoint instead of restarting. For more information about checkpointing, see Use Checkpoints in Amazon SageMaker (p. 600).

**Note**

Unless your training job will complete quickly, we recommend you use checkpointing with managed spot training. SageMaker built-in algorithms and marketplace algorithms that do not checkpoint are currently limited to a `MaxWaitTimeInSeconds` of 3600 seconds (60 minutes).

**Topics**

- Using Managed Spot Training (p. 599)
- Managed Spot Training Lifecycle (p. 600)

---

**Using Managed Spot Training**

To use managed spot training, create a training job. Set `EnableManagedSpotTraining` to `True` and specify the `MaxWaitTimeInSeconds`. `MaxWaitTimeInSeconds` must be larger than `MaxRuntimeInSeconds`. For more information about creating a training job, see CreateTrainingJob (p. 931).
You can calculate the savings from using managed spot training using the formula \(1 - \frac{\text{BillableTimeInSeconds}}{\text{TrainingTimeInSeconds}} \times 100\). For example, if BillableTimeInSeconds is 100 and TrainingTimeInSeconds is 500, the savings is 80%.

Managed Spot Training Lifecycle

You can monitor a training job using TrainingJobStatus and SecondaryStatus returned by DescribeTrainingJob (p. 1066). The list below shows how TrainingJobStatus and SecondaryStatus values change depending on the training scenario:

- **Spot instances acquired with no interruption during training**
  1. InProgress: Starting ↠ Downloading ↠ Training ↠ Uploading

- **Spot instances interrupted once. Later, enough spot instances were acquired to finish the training job.**
  1. InProgress: Starting ↠ Downloading ↠ Training ↠ Interrupted ↠ Starting ↠ Downloading ↠ Training ↠ Uploading

- **Spot instances interrupted twice and MaxWaitTimeInSeconds exceeded.**
  1. InProgress: Starting ↠ Downloading ↠ Training ↠ Interrupted ↠ Starting ↠ Downloading ↠ Training ↠ Interrupted ↠ Downloading ↠ Training
  2. Stopping: Stopping
  3. Stopped: MaxWaitTimeExceeded

- **Spot instances were never launched.**
  1. InProgress: Starting
  2. Stopping: Stopping
  3. Stopped: MaxWaitTimeExceeded

Use Checkpoints in Amazon SageMaker

A checkpoint is a snapshot of the state of the model. They can be used with Managed Spot Training. If a training job is interrupted, a snapshot can be used to resume from a previously saved point. This can save training time.

Snapshots are saved to an Amazon S3 location you specify. You can configure the local path to use for snapshots or use the default. When a training job is interrupted, Amazon SageMaker copies the training data to Amazon S3. When the training job is restarted, the checkpoint data is copied to the local path. It can be used to resume at the checkpoint.

To enable checkpoints, provide an Amazon S3 location. You can optionally provide a local path and choose to use a shared folder. The default local path is /opt/ml/checkpoints/. For more information, see CreateTrainingJob (p. 931)

Provide Dataset Metadata to Training Jobs with an Augmented Manifest File

To classify data into different groupings, you train a model by using a dataset and metadata that act as labels. To include metadata with your dataset in a training job, use an augmented manifest file. When using an augmented manifest file, your dataset must be stored in Amazon Simple Storage Service (Amazon S3) and you must configure your training job to use dataset stored there. You specify the location and format of this dataset for one or more Channel (p. 1310). Augmented manifests can only
Augmented Manifest File format

When specifying a channel's parameters, you specify a path to the file, called a S3Uri. Amazon SageMaker interprets this URI based on the specified S3DataType in S3DataSource (p. 1501). The AugmentedManifestFile option defines a manifest format that includes metadata with the input data. Using an augmented manifest file is an alternative to preprocessing when you have labeled data. For training jobs using labeled data, you typically need to preprocess the dataset to combine input data with metadata before training. If your training dataset is large, preprocessing can be time consuming and expensive.

Augmented Manifest File Format

An augmented manifest file must be formatted in JSON Lines format. In JSON Lines format, each line in the file is a complete JSON object followed by a newline separator.

During training, Amazon SageMaker parses each JSON line and sends some or all of its attributes on to the training algorithm. You specify which attribute contents to pass and the order in which to pass them with the AttributeNames parameter of the CreateTrainingJob (p. 931) API. The AttributeNames parameter is an ordered list of attribute names that Amazon SageMaker looks for in the JSON object to use as training input.

For example, if you list ["line", "book"] for AttributeNames, the input data must include the attribute names of line and book in the specified order. For this example, the following augmented manifest file content is valid:

```
{"author": "Herman Melville", "line": "Call me Ishmael", "book": "Moby Dick"}
{"line": "It was love at first sight.", "author": "Joseph Heller", "book": "Catch-22"}
```

Amazon SageMaker ignores unlisted attribute names even if they precede, follow, or are in between listed attributes.

When using augmented manifest files, observe the following guidelines:

- The order of the attributes listed in the AttributeNames parameter determines the order of the attributes passed to the algorithm in the training job.
- The listed AttributeNames can be a subset of all of the attributes in the JSON line. Amazon SageMaker ignores unlisted attributes in the file.
- You can specify any type of data allowed by the JSON format in AttributeNames, including text, numerical, data arrays, or objects.
- To include an S3 URI as an attribute name, add the suffix -ref to it.

If an attribute name contains the suffix -ref, the attribute's value must be an S3 URI to a data file that is accessible to the training job. For example, if AttributeNames contains ["image-ref", "is-a-cat"], a valid augmented manifest file might contain these lines:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "is-a-cat": 1}
{"image-ref": "s3://mybucket/sample02/image2.jpg", "is-a-cat": 0}
```

For the first line of this manifest, Amazon SageMaker retrieves the contents of the S3 object s3://mybucket/foo/image1.jpg and streams it to the algorithm for training. The second line is the string representation of the is-a-cat attribute "1", which is followed by the contents of the second line.

To create an augmented manifest file, use Amazon SageMaker Ground Truth to create a labeling job. For more information, see Output Data (p. 75).
Stream Augmented Manifest File Data

Augmented manifest files are supported only for channels using Pipe input mode. For each channel, the data is extracted from its augmented manifest file and streamed (in order) to the algorithm through the channel’s named pipe. Pipe mode uses the first in first out (FIFO) method, so records are processed in the order in which they are queued. For information about Pipe input mode, see InputMode.

Attribute names with a "-ref" suffix point to preformatted binary data. In some cases, the algorithm knows how to parse the data. In other cases, you might need to wrap the data so that records are delimited for the algorithm. If the algorithm is compatible with RecordIO-formatted data, specifying RecordIO for RecordWrapperType solves this issue. If the algorithm is not compatible with RecordIO format, specify None for RecordWrapperType and make sure that your data is parsed correctly for your algorithm. Using the ["image-ref", "is-a-cat"] example, if you use RecordIO wrapping, the following stream of data is sent to the queue:

```plaintext
recordio_formatted(s3://mybucket/foo/image1.jpg)recordio_formatted("1")recordio_formatted(s3://mybucket/bar/image2.jpg)recordio_formatted("0")
```

Images that aren't wrapped with RecordIO format, are streamed with the corresponding is-a-cat attribute value as one record. This can cause a problem because the algorithm might not delimit the images and attributes correctly.

With augmented manifest files and Pipe mode in general, size limits of the EBS volume do not apply. This includes settings that otherwise must be within the EBS volume size limit such as S3DataDistributionType. For more information about Pipe mode and how to use it, see Using Your Own Training Algorithms - Input Data Configuration.

Use an Augmented Manifest File (Console)

To complete this procedure, you need:

- The URL of the S3 bucket where you’ve stored the augmented manifest file.
- To store the data that is listed in the augmented manifest file in an S3 bucket.
- The URL of the S3 bucket where you want to store the output of the job.

To use an augmented manifest file in a training job (console)

2. In the navigation pane, choose Training, then choose Training jobs.
3. Choose Create training job.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. It can have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about supported built-in algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 220). If you want to use a custom algorithm, make sure that it is compatible with Pipe mode.
6. (Optional) For Resource configuration, either accept the default values or, to reduce computation time, increase the resource consumption.
   a. (Optional) For Instance type, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
   b. For Instance count, use the default, 1.
   c. (Optional) For Additional volume per instance (GB), choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
   a. For **Channel name**, either accept the default (`train`) or enter a more meaningful name, such as `training-augmented-manifest-file`.
   b. For **InputMode**, choose `Pipe`.
   c. For **S3 data distribution type**, choose `FullyReplicated`. When training incrementally, fully replicating causes each ML compute instance to use a complete copy of the expanded dataset. For neural-based algorithms, such as Neural Topic Model (NTM) Algorithm (p. 343), choose `ShardedByS3Key`.
   d. If the data specified in the augmented manifest file is uncompressed, set the **Compression type** to `None`. If the data is compressed using gzip, set it to `Gzip`.
   e. (Optional) For **Content type**, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.
   f. For **Record wrapper**, if the dataset specified in the augmented manifest file is saved in RecordIO format, choose `RecordIO`. If your dataset is not saved as a RecordIO-formatted file, choose `None`.
   g. For **S3 data type**, choose `AugmentedManifestFile`.
   h. For **S3 location**, provide the path to the bucket where you stored the augmented manifest file.
   i. For **AugmentedManifestFile attribute names**, specify the name of an attribute that you want to use. The attribute name must be present within the augmented manifest file, and is case-sensitive.
   j. (Optional) To add more attribute names, choose **Add row** and specify another attribute name for each attribute.
   k. (Optional) To adjust the order of attribute names, choose the up or down buttons next to the names. When using an augmented manifest file, the order of the specified attribute names is important.
   l. Choose **Done**.

8. For **Output data configuration**, provide the following information:
   a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
   b. (Optional) You can use your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. For **Encryption key**, provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

9. (Optional) For **Tags**, add one or more tags to the training job. A tag is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with `Project` as a key and a value that refers to a project that is related to the training job, such as `Home value forecasts`.

10. Choose **Create training job**. Amazon SageMaker creates and runs the training job.

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 34).

### Use an Augmented Manifest File (API)

The following shows how to train a model with an augmented manifest file using the Amazon SageMaker high-level Python library:

```python
# Create a model object set to using "Pipe" mode.
model = sagemaker.estimator.Estimator(training_image, role,
           train_instance_count=1,
           ...)  # Other parameters as needed
```

603
## Use an Augmented Manifest File (API)

Train a model using an augmented manifest file and the SageMaker Python SDK.

```python
train_instance_type='ml.p3.2xlarge',
train_volume_size = 50,
train_max_run = 360000,
input_mode = 'Pipe',
output_path=s3_output_location,
sagemaker_session=session)

# Create a train data channel with S3_data_type as 'AugmentedManifestFile' and attribute names.
train_data = sagemaker.session.s3_input(your_augmented_manifest_file,
distribution='FullyReplicated',
content_type='image/jpeg',
s3_data_type='AugmentedManifestFile',
attribute_names=['source-ref', 'annotations'])

data_channels = {'train': train_data}

# Train a model.
model.fit(inputs=data_channels, logs=True)
```

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for S3 output path in the Output data configuration field. To deploy the model to get predictions, see Step 6: Deploy the Model to Amazon SageMaker (p. 34).
Deploy Models

After you build and train your models, you can deploy them to get predictions in one of two ways:

- To set up a persistent endpoint to get predictions from your models, use Amazon SageMaker hosting services. For an overview on deploying a single model or multiple models with Amazon SageMaker hosting services, see Deploy a Model on Amazon SageMaker Hosting Services (p. 8).
- To get predictions for an entire dataset, use Amazon SageMaker batch transform. For an overview on deploying a model with Amazon SageMaker batch transform, see Get Inferences for an Entire Dataset with Batch Transform (p. 11).

Prerequisites

These topics assume that you have built and trained one or more machine learning models and are ready to deploy them. If you are new to Amazon SageMaker and have not completed these prerequisite tasks, work through the steps in the Get Started with Amazon SageMaker (p. 20) tutorial to familiarize yourself with an example of how Amazon SageMaker manages the data science process and how it handles model deployment. For more information about building a model, see Build Models (p. 196). For information about training a model, see Train Models (p. 504).

What do you want to do?

Amazon SageMaker provides features to manage resources and optimize inference performance when deploying machine learning models. For guidance on using inference pipelines, compiling and deploying models with Neo, Elastic Inference, and automatic model scaling, see the following topics.

- To manage data processing and real-time predictions or to process batch transforms in a pipeline, see Deploy an Inference Pipeline (p. 642).
- To train TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models once and optimize them to deploy on ARM, Intel, and Nvidia processors, see Compile and Deploy Models with Amazon SageMaker Neo (p. 663).
- To preprocess entire datasets quickly or to get inferences from a trained model for large datasets when you don't need a persistent endpoint, see Use Batch Transform (p. 656).
- To speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models using a GPU instance for your endpoint, see Use Amazon SageMaker Elastic Inference (EI) (p. 683).
- To dynamically adjust the number of instances provisioned in response to changes in your workload, see Automatically Scale Amazon SageMaker Models (p. 694).
- To create an endpoint that can host multiple models using a shared serving container, see Host Multiple Models with Multi-Model Endpoints (p. 606).

Manage Model Deployments

For guidance on managing model deployments, including monitoring, troubleshooting, and best practices, and for information on storage associated with inference hosting instances:
Deploy Your Own Inference Code

For developers that need more advanced guidance on how to run your own inference code:

- To run your own inference code hosting services, see Use Your Own Inference Code with Hosting Services (p. 480).
- To run your own inference code for batch transforms, see Use Your Own Inference Code with Batch Transform (p. 483).

Guide to Amazon SageMaker

What Is Amazon SageMaker? (p. 1)

Topics

- Host Multiple Models with Multi-Model Endpoints (p. 606)
- Amazon SageMaker Model Monitor (p. 616)
- Deploy an Inference Pipeline (p. 642)
- Use Batch Transform (p. 656)
- Compile and Deploy Models with Amazon SageMaker Neo (p. 663)
- Use Amazon SageMaker Elastic Inference (EI) (p. 683)
- Automatically Scale Amazon SageMaker Models (p. 694)
- Troubleshoot Amazon SageMaker Model Deployments (p. 708)
- Deployment Best Practices (p. 709)
- Host Instance Storage Volumes (p. 710)

Host Multiple Models with Multi-Model Endpoints

To create an endpoint that can host multiple models, use multi-model endpoints. Multi-model endpoints use a shared serving container enabled to host multiple models. This reduces hosting costs by improving endpoint utilization compared with using single model endpoints. It also reduces deployment overhead because Amazon SageMaker manages loading models in memory and scaling the models based on the traffic patterns to them. Multi-model endpoints provides a scalable and cost-effective solution to deploying large numbers of models.

Multi-model endpoints enable time-sharing of memory resources across your models. This works best when the models are fairly similar in size and invocation latency. When this is the case, multi-model endpoints can effectively use instances across all models. If you have models that have significantly higher transactions per second (TPS) or latency requirements, we recommend hosting them on dedicated endpoints. Multi-model endpoints are also well suited to cases that can tolerate occasional cold-start-related latency penalties that occur when invoking infrequently used models.
You can use multi-model endpoints with custom-built containers by integrating the Multi Model Server library. Multi-model endpoints also support A/B testing, and they work with Auto Scaling, and AWS PrivateLink. We don't support using MultiModel-enabled containers with serial inference pipelines and Amazon Elastic Inference (EIA).

Topics
- Sample Notebooks for Multi-Model Endpoints (p. 607)
- How Multi-Model Endpoints Work (p. 607)
- Multi-Model Endpoint Security (p. 607)
- Instance Recommendations for Multi-Model Endpoint Deployments (p. 608)
- CloudWatch Metrics for Multi-Model Endpoint Deployments (p. 609)
- Create a Multi-Model Endpoint (p. 609)
- Build Your Own Container with Multi Model Server (p. 612)
- Invoke a Multi-Model Endpoint (p. 615)
- Add or Remove Models (p. 615)

Sample Notebooks for Multi-Model Endpoints

For a sample notebook that uses Amazon SageMaker to deploy multiple XGBoost models to an endpoint, see the Multi-Model Endpoint XGBoost Sample Notebook. For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in Amazon SageMaker, see the Multi-Model Endpoint BYOC Sample Notebook. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After you've created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The Multi-Model Endpoint notebook is located in the ADVANCED FUNCTIONALITY section. To open a notebook, choose its Use tab and choose Create copy.

How Multi-Model Endpoints Work

Amazon SageMaker manages the lifecycle of the models in-memory for multi-model endpoints. Instead of downloading all models to the container from Amazon Simple Storage Service (Amazon S3) when you create the endpoint, a multi-model endpoint dynamically loads models from S3 when they are invoked. When Amazon SageMaker receives an invocation request for a particular model, it routes the request to an instance behind the endpoint, downloads the model from S3 to that instance, and loads the model to the container's memory. If a model is already loaded into memory, its invocation is fast because Amazon SageMaker doesn't need to download and load it.

Adding models to, and deleting them from, a multi-model endpoint doesn't require updating the endpoint itself. To add a model, you upload it to the S3 bucket and start invoking it. To delete a model, stop sending requests and delete it from the S3 bucket. The Amazon SageMaker platform provides multi-model endpoint capability in a serving container, so you don’t need code changes to use it.

When UpdateEndpoint (p. 1233) is performed on a multi-model endpoint, your InvokeEndpoint requests may experience higher latencies as traffic is directed to the instances in the updated endpoint.

Multi-Model Endpoint Security

Models and data in a multi-model endpoint are co-located on instance storage volume and in container memory. All instances for Amazon SageMaker endpoints run on a single tenant container that you own. Only your models can run on your multi-model endpoint. It's your responsibility to manage the mapping of requests to models and to provide access for users to the correct target models. Amazon SageMaker
uses IAM roles to provide IAM identity-based policies that you use to specify allowed or denied actions and resources and the conditions under which actions are allowed or denied.

An IAM principal with InvokeEndpoint permissions on a multi-model endpoint can invoke any model at the address of the S3 prefix defined in the CreateModel operation, provided that the IAM Execution Role defined in operation has permissions to download the model. If you need to restrict InvokeEndpoint access to a limited set of models in S3, you can create multi-model endpoints with more restrictive S3 prefixes. For more information about how Amazon SageMaker uses roles to manage access to endpoints and perform operations on your behalf, see Amazon SageMaker Roles (p. 758).

Your customers might also have certain data isolation requirements dictated by their own compliance requirements that can be satisfied using IAM identities.

**Instance Recommendations for Multi-Model Endpoint Deployments**

There are several items to consider when selecting a SageMaker ML instance type for a multi-model endpoint. Provision sufficient Amazon Elastic Block Store (Amazon EBS) capacity for all of the models that need to be served. Balance performance (minimize cold starts) and cost (don't over-provision instance capacity). For information about the size of the storage volume that Amazon SageMaker attaches for each instance type for an endpoint and for a multi-model endpoint, see Host Instance Storage Volumes (p. 710). For a container configured to run in MultiModel mode, the storage volume provisioned for its instances has more memory. This allows more models to be cached on the instance storage volume.

When choosing an Amazon SageMaker ML instance type, consider the following:

- The traffic distribution (access patterns) to the models that you want to host behind the multi-model endpoint, along with the model size (how many models could be loaded in memory on the instance).
- Think of the amount of memory on an instance as the cache space for models to be loaded. Think of the number of vCPUs as the concurrency limit to perform inference on the loaded models (assuming that invoking a model is bound to CPU).
- A higher amount of instance memory allows you to have more models loaded and ready to serve inference requests. You don't need to waste time loading the model.
- A higher amount of vCPUs allows you to invoke more unique models concurrently (again assuming that inference is bound to CPU).
- Have some "slack" memory available so that unused models can be unloaded, and especially for multi-model endpoints with multiple instances. If an instance or an Availability Zone fails, the models on those instances will be rerouted to other instances behind the endpoint.
- Tolerance to loading/downloading times:
  - d instance type families (for example, m5d, c5d, or r5d) come with an NVMe (non-volatile memory express) SSD, which offers high I/O performance and might reduce the time it takes to download models to the storage volume and for the container to load the model from the storage volume.
  - Because d instance types come with an NVMe SSD storage, Amazon SageMaker does not attach an Amazon EBS storage volume to these ML compute instances that hosts the multi-model endpoint.

In some cases, you might opt to reduce costs by choosing an instance type that can't hold all of the targeted models in memory at once. Amazon SageMaker dynamically unload models when it runs out of memory to make room for a newly targeted model. For infrequently requested models, you are going to pay a price with the dynamic load latency. In cases with more stringent latency needs, you might opt for larger instance types or more instances. Investing time up front for proper performance testing and analysis will pay great dividends in successful production deployments.

You can use the Average statistic of the ModelCacheHit metric to monitor the ratio of requests where the model is already loaded. You can use the SampleCount statistic for the ModelUnloadingTime metric.
metric to monitor the number of unload requests sent to the container during a time period. If models are unloaded too frequently (an indicator of thrashing, where models are being unloaded and loaded again because there is insufficient cache space for the working set of models), consider using a larger instance type with more memory or increasing the number of instances behind the multi-model endpoint. For multi-model endpoints with multiple instances, be aware that a model might be loaded on more than 1 instance.

Amazon SageMaker multi-model endpoints fully supports Auto Scaling, which manages replicas of models to ensure models scale based on traffic patterns. We recommend that you configure your multi-model endpoint and the size of your instances by considering all of the above and also set up auto scaling for your endpoint. The invocation rates used to trigger an auto-scale event is based on the aggregate set of predictions across the full set of models served by the endpoint.

CloudWatch Metrics for Multi-Model Endpoint Deployments

Amazon SageMaker provides metrics for endpoints so you can monitor the cache hit rate, the number of models loaded, and the model wait times for loading, downloading, and uploading at a multi-model endpoint. For information, see Multi-Model Endpoint Model Loading Metrics and Multi-Model Endpoint Model Instance Metrics in Monitor Amazon SageMaker with Amazon CloudWatch (p. 712). Per-model metrics aren't supported.

Create a Multi-Model Endpoint

You can use the AWS SDK for Python (Boto) or the Amazon SageMaker to create a multi-model endpoint.

Topics
- Create a Multi-Model Endpoint (AWS SDK for Python (Boto)) (p. 609)
- Create a Multi-Model Endpoint (Console) (p. 610)

Create a Multi-Model Endpoint (AWS SDK for Python (Boto))

You create a multi-model endpoint using the Amazon SageMaker CreateModel, CreateEndpointConfig, and CreateEndpoint APIs just as you would create a single model endpoint, but with two changes. When defining the container, you need to pass a new Mode parameter value, MultiModel. You also need to pass the ModelDataUrl field that specifies the prefix in Amazon S3 where the model artifacts are located, instead of the path to a single model artifact, as you would when deploying a single model.

For a sample notebook that uses Amazon SageMaker to deploy multiple XGBoost models to an endpoint, see Multi-Model Endpoint XGBoost Sample Notebook.

The following procedure outlines the key steps used in that sample to create a multi-model endpoint.

To deploy the model (AWS SDK for Python (Boto 3))

1. Get a container whose image supports deploying models.

```python
container = { 'Image': '123456789012.dkr.ecr.us-east-1.amazonaws.com/myimage:mytag',
             'ModelDataUrl': 's3://my-bucket/path/to/artifacts/',
             'Mode': 'MultiModel' }
```

2. Create the model that uses this container.
response = sm_client.create_model(  
    ModelName = 'my-multi-model-name',  
    ExecutionRoleArn = role,  
    Containers = [container])

3. Configure the multi-model endpoint for the model. We recommend configuring your endpoints with at least two instances. This allows Amazon SageMaker to provide a highly available set of predictions across multiple Availability Zones for the models.

response = sm_client.create_endpoint_config(  
    EndpointConfigName = 'my-epc',  
    ProductionVariants=[{  
        'InstanceType': 'ml.m4.xlarge',  
        'InitialInstanceCount': 2,  
        'InitialVariantWeight': 1,  
        'ModelName': 'my-multi-model-name',  
        'VariantName': 'AllTraffic'}])

4. Create the multi-model endpoint using the EndpointName and EndpointConfigName parameters.

response = sm_client.create_endpoint(  
    EndpointName = 'my-endpoint',  
    EndpointConfigName = 'my-epc')

---

Create a Multi-Model Endpoint (Console)

**To create a multi-model endpoint (console)**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Model, and then from the Inference group, choose Create model.
3. For Model name, enter a name.
4. For IAM role, choose or create an IAM role that has the AmazonSageMakerFullAccess IAM policy attached.
5. In the Container definition section, for Provide model artifacts and inference image options choose Use multiple models.
Create model

To deploy a model to Amazon SageMaker, first create the model by providing the location of your inference code. See Deploying a Model on Amazon SageMaker Hosting Services for more information.

Model settings

Model name

mml-test-model

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique across your account in an AWS Region.

IAM role

Amazon SageMaker requires permissions to call other services on your behalf. Choose a role or let us create a role that has the AmazonSageMakerFullAccess IAM policy attached.

AmazonSageMaker-ExecutionRole- XXXXXXXXXXXXXXXXXXXX

Container definition 1

- Container input options
  - Provide model artifacts and inference image location
  - Provide model artifacts and inference image options

- Use a single model
  - Use this to host a single model in this container.

- Use multiple models
  - Use this to host multiple models in this container.

Location of inference code image

Type the registry path where the inference code image is stored in Amazon ECR.

123456789012.dkr.ecr.us-east-1.amazonaws.com/myimage:mytag

Location of model artifacts

Type the URL where model artifacts are stored in S3.

s3://my-bucket/path/to/artifacts/
6. Choose Create model.
7. Deploy your multi-model endpoint as you would a single model endpoint.

Build Your Own Container with Multi Model Server

Custom Elastic Container Registry (ECR) images deployed in Amazon SageMaker are expected to adhere to the basic contract described in Use Your Own Inference Code with Hosting Services (p. 480) that govern how Amazon SageMaker interacts with a Docker container that runs your own inference code. For a container to be capable of loading and serving multiple models concurrently, there are additional APIs and behaviors that must be followed. This additional contract includes new APIs to load, list, get, and unload models, and a different API to invoke models. There are also different behaviors for error scenarios that the APIs need to abide by. To indicate that the container complies with the additional requirements, you can add the following command to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.multi-models=true
```

Amazon SageMaker also injects an environment variable into the container

```
SAGEMAKER_MULTI_MODEL=true
```

To help you implement these requirements for a custom container, two libraries are available:

- **Multi Model Server** is an open source framework for serving machine learning models that can be installed in containers to provide the front end that fulfills the requirements for the new multi-model endpoint container APIs. It provides the HTTP front end and model management capabilities required by multi-model endpoints to host multiple models within a single container, load models into and unload models out of the container dynamically, and performs inference on a specified loaded model. It also provides a pluggable backend that supports a pluggable custom backend handler where you can implement your own algorithm.

- **Amazon SageMaker Inference Toolkit** is a library that bootstraps Multi Model Server with a configuration and settings that make it compatible with Amazon SageMaker multi-model endpoints. It also allows you to tweak important performance parameters, such as the number of workers per model, depending on the needs of your scenario.

For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in Amazon SageMaker, see the Multi-Model Endpoint BYOC Sample Notebook.

**Contract for Custom Containers to Serve Multiple Model**

To handle multiple models, your container must support a set of APIs that enable the Amazon SageMaker platform to communicate with the container for loading, listing, getting, and unloading models as required. The model_name is used in the new set of APIs as the key input parameter. The customer container is expected to keep track of the loaded models using model_name as the mapping key. Also, the model_name is an opaque identifier and is not necessarily the value of the TargetModel parameter passed into the InvokeEndpoint API. The original TargetModel value in the InvokeEndpoint request is passed to container in the APIs as a X-Amzn-Target-Model header that can be used for logging purposes.

**Topics**

- LOAD MODEL API (p. 613)
- LIST MODEL API (p. 613)
- GET MODEL API (p. 614)
LOAD MODEL API

Instructs the container to load a particular model present in the `url` field of the body into the memory of the customer container and to keep track of it with the assigned `model_name`. After a model is loaded, the container should be ready to serve inference requests using this `model_name`.

```plaintext
POST /models HTTP/1.1
Content-Type: application/json
Accept: application/json

{
    "model_name" : "model_name",
    "url" : "/opt/ml/models/model_name/model",
}
```

**Note**
If `model_name` is already loaded, this API should return 409. Any time a model cannot be loaded due to lack of memory or to any other resource, this API should return a 507 HTTP status code to Amazon SageMaker, which then initiates unloading unused models to reclaim.

LIST MODEL API

Returns the list of models loaded into the memory of the customer container.

```plaintext
GET /models HTTP/1.1
Accept: application/json

Response =
{
    "models": [
        {
            "modelName" : "model_name",
            "modelUrl" : "/opt/ml/models/model_name/model",
        },
        {
            "modelName" : "model_name",
            "modelUrl" : "/opt/ml/models/model_name/model",
        },
        ....
    ]
}
```

This API also supports pagination.

```plaintext
GET /models HTTP/1.1
Accept: application/json

Response =
{
    "models": [
        {
            "modelName" : "model_name",
            "modelUrl" : "/opt/ml/models/model_name/model",
        },
        {
```

613
"modelName" : "{model_name}",
"modelUrl" : "/opt/ml/models/{model_name}/model",
},
,...
]
}

Amazon SageMaker can initially call the List Models API without providing a value for next_page_token. If a nextPageToken field is returned as part of the response, it will be provided as the value for next_page_token in a subsequent List Models call. If a nextPageToken is not returned, it means that there are no more models to return.

**GET MODEL API**

This is a simple read API on the model_name entity.

```
GET /models/{model_name} HTTP/1.1
Accept: application/json
{
    "modelName" : "{model_name}",
    "modelUrl" : "/opt/ml/models/{model_name}/model",
}
```

**Note**

If model_name is not loaded, this API should return 404.

**UNLOAD MODEL API**

Instructs the Amazon SageMaker platform to instruct the customer container to unload a model from memory. This initiates the eviction of a candidate model as determined by the platform when starting the process of loading a new model. The resources provisioned to model_name should be reclaimed by the container when this API returns a response.

```
DELETE /models/{model_name}
```

**Note**

If model_name is not loaded, this API should return 404.

**INVOKE MODEL API**

Makes a prediction request from the particular model_name supplied. The Amazon SageMaker Runtime InvokeEndpoint request supports X-Amzn-Target-Model as a new header that takes the relative path of the model specified for invocation. The Amazon SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the CreateModel API call with the relative path of the model.

```
POST /models/{model_name}/invoke HTTP/1.1
Content-Type: Content-type
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes:CustomAttributes
X-Amzn-SageMaker-Target-Model: [relativePath]/{artifactName}.tar.gz
```

**Note**

If model_name is not loaded, this API should return 404.
Invoke a Multi-Model Endpoint

To invoke a multi-model endpoint, use the `InvokeEndpoint` from the Amazon SageMaker Runtime just as you would invoke a single model endpoint, with one change. Pass a new `TargetModel` parameter that specifies which of the models at the endpoint to target. The Amazon SageMaker Runtime `InvokeEndpoint` request supports `X-Amzn-Target-Model` as a new header that takes the relative path of the model specified for invocation. The Amazon SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the `CreateModel` API call with the relative path of the model.

The following example prediction request uses the AWS SDK for Python (Boto 3) in the sample notebook.

```python
response = runtime_sm_client.invoke_endpoint(
    EndpointName = 'my-endpoint',
    ContentType  = 'text/csv',
    TargetModel  = 'Houston_TX.tar.gz',
    Body         = body)
```

The multi-model endpoint dynamically loads target models as needed. You can observe this when running the MME Sample Notebook as it iterates through random invocations against multiple target models hosted behind a single endpoint. The first request against a given model takes longer because the model has to be downloaded from Amazon Simple Storage Service (Amazon S3) and loaded into memory. (This is called a cold start.) Subsequent calls finish faster because there's no additional overhead after the model has loaded.

**Note**
Invoking multi-model endpoints using the Amazon SageMaker Python SDK isn't supported.

Add or Remove Models

You can deploy additional models to a multi-model endpoint and invoke them through that endpoint immediately. When adding a new model, you don't need to update or bring down the endpoint, so you avoid the cost of creating and running a separate endpoint for each new model.

Amazon SageMaker unloads unused models from the container when the instance is reaching memory capacity and more models need to be downloaded into the container. Amazon SageMaker also deletes unused model artifacts from the instance storage volume when the volume is reaching capacity and new models need to be downloaded. The first invocation to a newly added model takes longer because the endpoint takes time to download the model from S3 to the container's memory in instance hosting the endpoint.

With the endpoint already running, copy a new set of model artifacts to the Amazon S3 location there you store your models.

```bash
# Add an AdditionalModel to the endpoint and exercise it
aws s3 cp AdditionalModel.tar.gz s3://my-bucket/path/to/artifacts/
```

**Important**
To update a model, proceed as you would when adding a new model. Use a new and unique name. Don't overwrite model artifacts in Amazon S3 because the old version of the model might still be loaded in the containers or on the storage volume of the instances on the endpoint. Invocations to the new model could then invoke the old version of the model.

Client applications can request predictions from the additional target model as soon as it is stored in S3.

```python
response = runtime_sm_client.invoke_endpoint(
```
Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor continuously monitors the quality of Amazon SageMaker machine learning models in production. It enables developers to set alerts for when there are deviations in the model quality, such as data drift. Early and pro-active detection of these deviations enables you to take corrective actions, such as retraining models, auditing upstream systems, or fixing data quality issues without having to monitor models manually or build additional tooling. You can use Model Monitor pre-built monitoring capabilities that do not require coding. You also have the flexibility to monitor models by coding to provide custom analysis.

How Model Monitor Works

Amazon SageMaker Model Monitor automatically monitors machine learning (ML) models in production and notifies you when data quality issues arise. ML models in production have to make predictions on real-life data that is not carefully curated like most training datasets. If the statistical nature of the data that your model receives while in production drifts away from the nature of the baseline data it was trained on, the model begins to lose accuracy in its predictions. Model Monitor uses rules to detect data drift and alerts you when it happens. The following figure shows how this process works.

To enable model monitoring, you take the following steps, which follow the path of the data through the various data collection, monitoring, and analysis processes.

- **Capture Data (p. 617)**: Enable the endpoint to capture data from incoming requests to a trained ML model and the resulting model predictions.
- **Create a Baseline (p. 619)**: Create a baseline from the dataset that was used to train the model. Compute baseline schema constraints and statistics for each feature using Deequ, an open source library built on Apache Spark, which is used to measure data quality in large datasets.
- **Schedule Monitoring Jobs (p. 620)**: Create a monitoring schedule specifying what data to collect, how often to collect it, how to analyze it, and which reports to produce.
Capture Data

- **Interpret Results (p. 623):** Inspect the reports, which compare the latest data with the baseline, and watch for any violations reported and for metrics and notifications from Amazon CloudWatch.

**Note**
Amazon SageMaker Model Monitor currently supports only endpoints that host a single model and does not support monitoring multi-model endpoints. For information on using multi-model endpoints, see Host Multiple Models with Multi-Model Endpoints (p. 606).

**Model Monitor Sample Notebooks**

For a sample notebook that takes you through the full end-to-end workflow for Model Monitor, see the Introduction to Amazon SageMaker Model Monitor.

For a sample notebook that enables the model monitoring experience for an existing endpoint, see the Enable Model Monitoring.

For a sample notebook that visualizes the statistics.json file for a selected execution in a monitoring schedule, see the Model Monitor Visualization.

For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). Once you have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, choose its Use tab and choose Create copy.

**Capture Data**

After you have created an endpoint, configure the permissions and paths to Amazon S3 locations for storing data, report, and processing code.

```python
import boto3
import re
import json
from sagemaker import get_execution_role, session

region = boto3.Session().region_name

role = get_execution_role()
print("RoleArn: {}".format(role))

# You can use a different bucket, but make sure the role you chose for this notebook has s3:PutObject permissions. This is the bucket into which the data is captured
bucket = session.Session(boto3.Session()).default_bucket()
print("Demo Bucket: {}".format(bucket))

prefix = 'sagemaker/DEMO-ModelMonitor'
data_capture_prefix = '{}{}'.format(prefix, data_capture_prefix)
s3_capture_upload_path = 's3:///{}/{}' .format(bucket, data_capture_prefix)
reports_prefix = '{}reports'.format(prefix)
s3_report_path = 's3:///{}/{}' .format(bucket, reports_prefix)

print("Capture path: {}" .format(s3_capture_upload_path))
print("Report path: {}" .format(s3_report_path))

Upload the pre-trained model to Amazon S3:
```
Enable data capture: You specify the capture option called DataCaptureConfig (p. 1326). You can capture the request payload, the response payload, or both with this configuration. The capture configuration applies to all variants.

```
from sagemaker.model_monitor import DataCaptureConfig

endpoint_name = 'your-pred-model-monitor-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config = DataCaptureConfig(
    enable_capture = True,
    sampling_percentage=100,
    destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                          instance_type='ml.m4.xlarge',
                          endpoint_name=endpoint_name,
                          data_capture_config=data_capture_config)
```

Invoke the deployed model: You can now send data to this endpoint to get inferences in real time. Because you enabled the data capture in the previous steps, the request and response payload, along with some additional metadata, is saved in the Amazon S3 location that you specified in DataCaptureConfig (p. 1326).

```
from sagemaker.predictor import RealTimePredictor
import time

predictor = RealTimePredictor(endpoint=endpoint_name, content_type='text/csv')

# get a subset of test data for a quick test
!head -120 test_data/test-dataset-input-cols.csv > test_data/test_sample.csv
print("Sending test traffic to the endpoint {}. Please wait...".format(endpoint_name))

with open('test_data/test_sample.csv', 'r') as f:
    for row in f:
        payload = row.rstrip('
')
        response = predictor.predict(data=payload)
        time.sleep(0.5)

print("Done!")
```

View captured data: List the data capture files stored in Amazon S3. Expect to see different files from different time periods, organized based on the hour when the invocation occurred.

```
s3_client = boto3.Session().client('s3')
current_endpoint_capture_prefix = '{}/{}'.format(data_capture_prefix, endpoint_name)
result = s3_client.list_objects(Bucket=bucket, Prefix=current_endpoint_capture_prefix)
capture_files = [capture_file.get("Key") for capture_file in result.get('Contents')]
print("Found Capture Files:")
print("\n".join(capture_files))
```

The format of the Amazon S3 path is:

```
s3://{destination-bucket-prefix}/{endpoint-name}/{variant-name}/yyyy/mm/dd/hh/
filename.jsonl
```
Create a Baseline

The baseline calculations of statistics and constraints are needed as a standard against which data drift and other data quality issues can be detected. Amazon SageMaker Model Monitor provides a built-in container that provides the ability to suggest the constraints automatically for CSV and flat JSON input. This `sagemaker-model-monitor-analyzer` container also provides you with a range of model monitoring capabilities, including constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with Deequ.

The training dataset that you used to trained the model is usually a good baseline dataset. The training dataset data schema and the inference dataset schema should exactly match (the number and order of the features). Note that the prediction/output column(s) are assumed to be the 1st column(s) in the training dataset. From the training dataset, you can ask Amazon SageMaker to suggest a set of baseline constraints and generate descriptive statistics to explore the data. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already have it in Amazon S3, you can point to it directly.

**Create a baseline from a training dataset:** When you have your training data ready and stored in Amazon S3, start a baseline processing job with `DefaultModelMonitor.suggest_baseline(..)` using the Amazon SageMaker Python SDK. This uses an Amazon SageMaker Model Monitor Pre-built Container (p. 622) that generates baseline statistics and suggests baseline constraints for the dataset and writes them to the `output_s3_uri` location that you specify.

```python
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri+'/training-dataset-with-header.csv',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

**Note**

If you provide the feature/column names in the training dataset as the 1st row and set the `header=True` option as in the code sample above, Amazon SageMaker uses the feature name in the constraints and statistics file.

The baseline statistics for the dataset are contained in the `statistics.json` file and the suggested baseline constraints are contained in the `constraints.json` file in the location you specify with `output_s3_uri`.

**Table: Output Files for Tabular Dataset Statistics and Constraints**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statistics.json</td>
<td>This file is expected to have columnar statistics for each feature in the dataset that is analyzed. See the schema for this file in the <a href="#">Schema for Statistics</a> section.</td>
</tr>
<tr>
<td>constraints.json</td>
<td>This file is expected to have the constraints on the features observed. See the schema for this file in the <a href="#">Schema for Constraints</a> section.</td>
</tr>
</tbody>
</table>
The Amazon SageMaker Python SDK provides convenience functions described to generate the baseline statistics and constraints. But if you want to call processing job directly for this purpose instead, you need to set the Environment map as in the following example.

```json
"Environment": {
    "dataset_format": "\"csv\": \{"header\": true",
    "dataset_source": "/opt/ml/processing/sm_input",
    "output_path": "/opt/ml/processing/sm_output",
    "publish_cloudwatch_metrics": "Disabled"
}
```

## Schedule Monitoring Jobs

Amazon SageMaker Model Monitor provides you the ability to continuously monitor the data collected from the endpoints on a schedule. You can create a monitoring schedule with the CreateMonitoringSchedule (p. 910) API with a predefined periodic interval. For example, every x hours (x can range from 1 to 23).

With a Monitoring Schedule, Amazon SageMaker can kick off processing jobs at a specified frequency to analyze the data collected during a given period. Amazon SageMaker provides a pre-built container for performing analysis on tabular datasets. In the processing job, Amazon SageMaker compares the dataset for the current analysis with the baseline statistics, constraints provided and generate a violations report. In addition, CloudWatch metrics are emitted for each feature under analysis. Alternatively, you could choose to bring your own container as outlined in the Bring Your Own Containers (p. 631) topic.

You can create a model monitoring schedule for the endpoint created earlier. Use the baseline resources (constraints and statistics) to compare against the real-time traffic. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already have it in Amazon S3, you can point to it directly.

```python
# copy over the training dataset to Amazon S3 (if you already have it in Amazon S3, you could reuse it)
baseline_prefix = prefix + '/baselining'
baseline_data_prefix = baseline_prefix + '/data'
baseline_results_prefix = baseline_prefix + '/results'
baseline_data_uri = 's3://{}/{}'.format(bucket, baseline_data_prefix)
baseline_results_uri = 's3://{}/{}'.format(bucket, baseline_results_prefix)
print('Baseline data uri: {}'.format(baseline_data_uri))
print('Baseline results uri: {}'.format(baseline_results_uri))

training_data_file = open("test_data/training-dataset-with-header.csv", 'rb')
s3_key = os.path.join(baseline_prefix, 'data', 'training-dataset-with-header.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(training_data_file)
```

Create a model monitoring schedule for the endpoint using the baseline constraints and statistics to compare against real-time traffic.

```python
from sagemaker.model_monitor import CronExpressionGenerator
from time import gmtime, strftime
```
mon_schedule_name = 'DEMO-xgb-churn-pred-model-monitor-schedule-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
my_default_monitor.create_monitoring_schedule(
    schedule_name=mon_schedule_name,
    endpoint_input=predictor.endpoint,
    post_analytics_processor_script=s3_code_postprocessor_uri,
    output_s3_uri=s3_report_path,
    statistics=my_default_monitor.baseline_statistics(),
    constraints=my_default_monitor.suggested_constraints(),
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
)

Describe and inspect the schedule: After you describe it, observe that the MonitoringScheduleStatus in MonitoringScheduleSummary (p. 1452) returned by the ListMonitoringSchedules (p. 1154) API changes to Scheduled.

desc_schedule_result = my_default_monitor.describe_schedule()
print('Schedule status: {}'.format(desc_schedule_result['MonitoringScheduleStatus']))

The cron Expression for Monitoring Schedule

To provide details for the monitoring schedule, use ScheduleConfig (p. 1503), which is a cron expression that describes details about the monitoring schedule.

Amazon SageMaker Model Monitor supports the following cron expressions:

- To set the job to start every hour:
  
  Hourly: cron(0 * ? * *)

- To run the job daily:
  
  cron(0 [00-23] ? * *)

For example, the following are valid cron expressions:

- Daily at 12 PM UTC: cron(0 12 ? * *)
- Daily at 12 AM UTC: cron(0 0 ? * *)

To support running every 6, 12 hours, Model Monitoring supports the following expression:

cron(0 [00-23]/[01-24] ? * *)

For example, the following are valid cron expressions:

- Every 12 hours, starting at 5 PM UTC: cron(0 17/12 ? * *)
- Every two hours, starting at 12 AM UTC: cron(0 0/2 ? * *)

Note

- Although the cron expression is set to start at 5 PM UTC, note that there could be a delay of 0-20 minutes from the actual requested time to run the execution.
- If you want to run on a daily schedule, don't provide this parameter. Amazon SageMaker picks a time to run every day
• Currently, Amazon SageMaker only supports hourly integer rates between 1 hour and 24 hours.

Amazon SageMaker Model Monitor Pre-built Container

Amazon SageMaker provides a built-in container sagemaker-model-monitor-analyzer that provides you with a range of model monitoring capabilities, including constraint suggestion, statistics generation, constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with Deequ. The prebuilt container for SageMaker Model Monitor can be accessed at:

<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-model-monitor-analyzer

For example: 159807026194.dkr.ecr.us-west-1.amazonaws.com/sagemaker-model-monitor-analyzer

The following table lists the supported values for account IDs and corresponding AWS Region names.

<table>
<thead>
<tr>
<th>ACCOUNT_ID</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>895015795356</td>
<td>eu-north-1</td>
</tr>
<tr>
<td>607024016150</td>
<td>me-south-1</td>
</tr>
<tr>
<td>126357580389</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>680080141114</td>
<td>us-east-2</td>
</tr>
<tr>
<td>777275614652</td>
<td>us-east-2</td>
</tr>
<tr>
<td>468650794304</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>048819808253</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>539772159869</td>
<td>sa-east-1</td>
</tr>
<tr>
<td>001633400207</td>
<td>ap-east-1</td>
</tr>
<tr>
<td>156813124566</td>
<td>us-east-1</td>
</tr>
<tr>
<td>709848358524</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>749857270468</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>574779866223</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>159807026194</td>
<td>us-west-2</td>
</tr>
<tr>
<td>890145073186</td>
<td>us-west-1</td>
</tr>
<tr>
<td>245545462676</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>563025443158</td>
<td>ap-southeast-2</td>
</tr>
<tr>
<td>536280801234</td>
<td>ca-central-1</td>
</tr>
</tbody>
</table>

To write your own analysis container, see the container contract described in Customize Monitoring (p. 629).
Interpret Results

Once you have run a baseline processing job and obtained statistics and constraint for your dataset, you can execute monitoring jobs that calculate statistics and list any violations encountered relative to the baseline constraints. Amazon CloudWatch metrics are also reported in your account by default. For information on viewing the results of monitoring in Amazon SageMaker Studio, see Visualize Results in Amazon SageMaker Studio (p. 627).

**List executions:** The schedule starts monitoring jobs at the specified intervals. The following code lists the latest five executions. If you are running this code after creating the hourly schedule, the executions might be empty, and you might have to wait until you cross the hour boundary (in UTC) to see the executions start. The following code includes the logic for waiting.

```python
mon_executions = my_default_monitor.list_executions()
print("We created a hourly schedule above and it will kick off executions ON the hour (plus 0 - 20 min buffer.\nWe will have to wait till we hit the hour...")
while len(mon_executions) == 0:
    print("Waiting for the 1st execution to happen...")
    time.sleep(60)
    mon_executions = my_default_monitor.list_executions()
```

**Inspect a specific execution:** In the previous cell, you picked up the latest completed or failed scheduled execution. You can explore what went right or wrong. The terminal states are:

- **Completed:** The monitoring execution completed and no issues were found in the violations report.
- **CompletedWithViolations:** The execution completed, but constraint violations were detected.
- **Failed:** The monitoring execution failed, possibly due to client error (for example, a role issues) or infrastructure issues. To identify the cause, see the `FailureReason` and `ExitMessage`.

```python
latest_execution = mon_executions[-1] # latest execution's index is -1, previous is -2 and so on..
time.sleep(60)
latest_execution.wait(logs=False)
print("Latest execution status: {}\n{}".format(latest_execution.describe()['ProcessingJobStatus']))
print("Latest execution result: {}\n{}".format(latest_execution.describe()['ExitMessage']))
latest_job = latest_execution.describe()
if (latest_job['ProcessingJobStatus'] != 'Completed'):
    print("====STOP==== \nNo completed executions to inspect further. Please wait till an execution completes or investigate previously reported failures.")
```

**List the generated reports:**

```python
report_uri=latest_execution.output.destination
print('Report Uri: {}\n{}'.format(report_uri))
```

```python
from urllib.parse import urlparse
s3uri = urlparse(report_uri)
report_bucket = s3uri.netloc
report_key = s3uri.path.lstrip('/
print('Report bucket: {}\n{}'.format(report_bucket))
print('Report key: {}\n{}'.format(report_key))
```
```python
s3_client = boto3.Session().client('s3')
result = s3_client.list_objects(Bucket=report_bucket, Prefix=report_key)
report_files = [report_file.get("Key") for report_file in result.get('Contents')]
print("Found Report Files:"
print("\n ".join(report_files))
```

**Violations report:** If there are violations compared to the baseline, they are generated in the violations report. List the violations.

```python
violations = my_default_monitor.latest_monitoring_constraintViolations()
pd.set_option('display.max_colwidth', -1)
constraints_df = pd.io.json.json_normalize(violations.body_dict["violations"])
constraints_df.head(10)
```

This applies only to datasets that contain tabular data. The following schema files specify the statistics calculated and the violations monitored for.

**Table: Output Files for Tabular Datasets**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>statistics.json</strong></td>
<td>Contains columnar statistics for each feature in the dataset that is analyzed. See the schema of this file in the next topic.</td>
</tr>
<tr>
<td><strong>constraints_violations.json</strong></td>
<td>Contains a list of violations found in this current set of data as compared to the baseline statistics and constraints file specified in the baseline_constraints and baseline_statistics paths.</td>
</tr>
</tbody>
</table>

The Amazon SageMaker Model Monitor Pre-built Container (p. 622) saves a set of Amazon CloudWatch metrics for each feature by default.

The container code can emit CloudWatch metrics in this location: /opt/ml/output/metrics/cloudwatch. The schema for these files is outlined in the following topics.

**Topics**
- Schema for Statistics (statistics.json file) (p. 624)
- Schema for Violations (constraint_violations.json file) (p. 626)
- CloudWatch Metrics (p. 627)
- Visualize Results in Amazon SageMaker Studio (p. 627)

**Schema for Statistics (statistics.json file)**

Amazon SageMaker Model Monitor pre-built container computes per column/feature statistics. The statistics are calculated for the baseline dataset and also for the current dataset that is being analyzed.

```json
{
    "version": 0,
    "dataset": {
        "item_count": number
    }
}
```
"features": [
{
"name": "feature-name",
"inferred_type": "Fractional" | "Integral",
"numerical_statistics": {
"common": {
"num_present": number,
"num_missing": number
},
"mean": number,
"sum": number,
"std_dev": number,
"min": number,
"max": number,
"distribution": {
"kll": {
"buckets": [
{
"lower_bound": number,
"upper_bound": number,
"count": number
}
],
"sketch": {
"parameters": {
"c": number,
"k": number
},
"data": [
[
num,
num,
num,
num
],
[
num,
num
]
]}
}&sketch
}&KLL
}&distribution
}&num_stats
},
{
"name": "feature-name",
"inferred_type": "String",
"string_statistics": {
"common": {
"num_present": number,
"num_missing": number
},
"distinct_count": number,
"distribution": {
"categorical": {
"buckets": [
{
"value": "string",
"count": number
}
]
}
}
}
Note the following:

- The prebuilt containers compute **KLL sketch** which is a compact quantiles sketch.
- By default, we materialize the distribution in 10 buckets. This is not currently configurable.

### Schema for Violations (constraint_violations.json file)

The violations file is generated as the output of a `MonitoringExecution`, which lists the results of evaluating the constraints (specified in the constraints.json file) against the current dataset that was analyzed. The Amazon SageMaker Model Monitor pre-built container provides the following violation checks.

```json
{
  "violations": [{
    "feature_name": "string",
    "constraint_check_type": "data_type_check",
    "completeness_check",
    "baseline_drift_check",
    "missing_column_check",
    "extra_column_check",
    "categorical_values_check",
    "description": "string"
  }]
}
```

**Table: Types of Violations Monitored**

<table>
<thead>
<tr>
<th>Violation Check Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_type_check</td>
<td>If the data types in the current execution are not the same as in the baseline dataset, this violation is flagged.</td>
</tr>
<tr>
<td></td>
<td>During the baseline step, the generated constraints suggest the inferred data type for each column. The <code>monitoring_config.datatype_check_threshold</code> parameter can be tuned to adjust the threshold on when it is flagged as a violation.</td>
</tr>
<tr>
<td>completeness_check</td>
<td>If the completeness (% of non-null items) observed in the current execution exceeds the threshold specified in completeness threshold specified per feature, this violation is flagged.</td>
</tr>
<tr>
<td></td>
<td>During the baseline step, the generated constraints suggest a completeness value.</td>
</tr>
<tr>
<td>baseline_drift_check</td>
<td>If the calculated distribution distance between the current and the baseline datasets exceeds the drift threshold, this violation is flagged.</td>
</tr>
</tbody>
</table>

---

626
Violation Check Type | Description
--- | ---
missing_column_check | If the number of columns in the current dataset is less than the number in the baseline dataset, this violation is flagged.
extra_column_check | If the number of columns in the current dataset is more than the number in the baseline, this violation is flagged.
categorical_values_check | If there are more unknown values in the current dataset than in the baseline dataset, this violation is flagged. This value is dictated by the threshold in monitoring_config.domain_content_threshold.

CloudWatch Metrics

**Built-in Amazon SageMaker Model Monitor container for CloudWatch metrics:** When the emit_metrics option is Enabled in the baseline constraints file, Amazon SageMaker emits these metrics for each feature/column observed in the dataset in the /aws/sagemaker/Endpoints/data-metric namespace with EndpointName and ScheduleName dimensions.

For numerical fields:

- Metric: Max → query for MetricName: `feature_data_{feature_name}`, Stat: Max
- Metric: Min → query for MetricName: `feature_data_{feature_name}`, Stat: Min
- Metric: Sum → query for MetricName: `feature_data_{feature_name}`, Stat: Sum
- Metric: SampleCount → query for MetricName: `feature_data_{feature_name}`, Stat: SampleCount
- Metric: Average → query for MetricName: `feature_data_{feature_name}`, Stat: Average

For both numerical and string fields:

- Metric: Completeness → query for MetricName: `feature_non_null_{feature_name}`, Stat: Sum
- Metric: Baseline Drift → query for MetricName: `feature_baseline_drift_{feature_name}`, Stat: Sum

**Visualize Results in Amazon SageMaker Studio**

You can also visualize the results of monitoring in Amazon SageMaker Studio. For information about the onboarding process for using Studio, see Onboard to Amazon SageMaker Studio (p. 16).

You can view monitoring results at your endpoints.
You can view the jobs being monitoring.

You can take a deep dive into each monitoring results for each job.
Advanced Topics

The following sections contain more advanced tasks that explain how to customize monitoring using preprocessing and postprocessing scripts, how to build your own container, and how to use CloudFormation to create a monitoring schedule.

Topics
- Customize Monitoring (p. 629)
- Create a Monitoring Schedule with an AWS CloudFormation Custom Resource (p. 638)

Customize Monitoring

In addition to using the built-in monitoring mechanisms, you can create your own custom monitoring schedules and procedures using preprocessing and postprocessing scripts or by using or building your own container.

Topics
- Preprocessing and Postprocessing (p. 629)
- Bring Your Own Containers (p. 631)

Preprocessing and Postprocessing

In addition to using the built-in mechanisms, you can extend the code with the preprocessing and postprocessing scripts.

Topics
- Postprocessing Script (p. 629)
- Preprocessing Script (p. 629)

Postprocessing Script

You can extend the code with the post processing script by following this contract.

```python
def postprocess_handler():
    print("Hello from post-proc script!")
```

Specify it as a path in Amazon Simple Storage Service (Amazon S3) in the `CreateMonitoringSchedule` request.

```
```

Preprocessing Script

The Amazon SageMaker Model Monitor container works only with tabular or flattened json structures. We provide a per-record preprocessor for some small changes required to transform the dataset. For example, if your output is an array [1.0, 2.1], you need to convert this into a flattened JSON, like {"prediction0": 1.0, "prediction1": 2.1"}. A sample implementation might look like the following.

```python
def preprocess_handler(inference_record):
    event_data = inference_record.event_data
    input_data = {}
    output_data = {}
```
input_data['feature0'] = random.randint(1, 3)
input_data['feature1'] = random.uniform(0, 1.6)
input_data['feature2'] = random.uniform(0, 1.6)
output_data['prediction0'] = random.uniform(1, 30)
return {**input_data, **output_data}

Specify it as a path in Amazon S3 in the CreateMonitoringSchedule request:

```
```

The structure of the inference_record is defined as follows.

```
KEY_EVENT_METADATA = "eventMetadata"
KEY_EVENT_METADATA_EVENT_ID = "eventId"
KEY_EVENT_METADATA_EVENT_TIME = "inferenceTime"
KEY_EVENT_METADATA_CUSTOM_ATTR = "customAttributes"

KEY_EVENTDATA = "captureData"
KEY_EVENTDATA_INPUT = "endpointInput"
KEY_EVENTDATA_OUTPUT = "endpointOutput"
KEY_EVENTDATA_ENCODING = "encoding"
KEY_EVENTDATA_DATA = "data"
KEY_EVENTDATA_OBSERVED_CONTENT_TYPE = "observedContentType"
KEY_EVENTDATA_MODE = "mode"

KEY_EVENT_VERSION = "eventVersion"

```

```
"captureData": {
  "endpointInput": {
    "observedContentType": "text/csv",
    "mode": "INPUT",
    "data": "132,25,113.2,96,269.9,107,,0,0,0,0,0,1,0,0,0,0,1",
    "encoding": "CSV"
  },
  "endpointOutput": {
    "observedContentType": "text/csv; charset=utf-8",
    "mode": "OUTPUT",
    "data": "0.01076381653547287",
    "encoding": "CSV"
  }
},
"eventMetadata": {
  "eventId": "feca1ab1-8025-47e3-8f6a-99e3fdd7b8d9",
  "inferenceTime": "2019-11-20T23:33:12Z"
},
"eventVersion": "0"
```
```
class EventConfig:
    def __init__(self, endpoint, variant, start_time, end_time):
        self.endpoint = endpoint
        self.variant = variant
        self.start_time = start_time
        self.end_time = end_time

class EventMetadata:
    def __init__(self, event_metadata_dict):
        self.event_id = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_ID, None)
self.event_time = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_TIME, None)
self.custom_attribute =

self.encoding = data_dict.get(KEY_EVENTDATA_ENCODING, None)
self.data = data_dict.get(KEY_EVENTDATA_DATA, None)
self.observedContentType = data_dict.get(KEY_EVENTDATA_OBSERVED_CONTENT_TYPE, None)
self.mode = data_dict.get(KEY_EVENTDATA_MODE, None)

ret = {
    KEY_EVENTDATA_ENCODING: self.encoding,
    KEY_EVENTDATA_DATA: self.data,
    KEY_EVENTDATA_OBSERVED_CONTENT_TYPE: self.observedContentType,
}
return ret

class CapturedData:
    def __init__(self, event_dict):
        self.event_metadata = None
        self.endpoint_input = None
        self.endpoint_output = None
        self.event_version = None
        self.event_dict = event_dict
        self._event_dict_postprocessed = False
        if KEY_EVENT_METADATA in event_dict:
            self.event_metadata = EventMetadata(event_dict[KEY_EVENT_METADATA])
        if KEY_EVENTDATA in event_dict:
            if KEY_EVENTDATA_INPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_input = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT])
            if KEY_EVENTDATA_OUTPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_output = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_OUTPUT])
        if KEY_EVENT_VERSION in event_dict:
            self.event_version = event_dict[KEY_EVENT_VERSION]

    def as_dict(self):
        if self._event_dict_postprocessed is True:
            return self.event_dict
        if KEY_EVENTDATA in self.event_dict:
            if KEY_EVENTDATA_INPUT in self.event_dict[KEY_EVENTDATA]:
                self.event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT] = self.endpoint_input.as_dict()
            if KEY_EVENTDATA_OUTPUT in self.event_dict[KEY_EVENTDATA]:
                self.event_dict[KEY_EVENTDATA][KEY_EVENTDATA_OUTPUT] = self.endpoint_output.as_dict()
        self._event_dict_postprocessed = True
        return self.event_dict

Bring Your Own Containers

Amazon SageMaker Model Monitor provides a prebuilt container with ability to analyze the data captured from Endpoints for tabular datasets. If you would like to bring your own container, Model Monitor provides extension points which you can leverage.

Under the hood, when you create a MonitoringSchedule, Amazon SageMaker Model Monitor ultimately kicks off processing jobs. Hence the container needs to be aware of the processing job contract documented in the Build Your Own Processing Container (p. 191) topic. Note that Amazon
SageMaker Model Monitor kicks off the processing job on your behalf per the schedule. While invoking, Model Monitor sets up additional environment variables for you so that your container has enough context to process the data for that particular execution of the scheduled monitoring. For additional information on container inputs, see the Container Contract Inputs (p. 632).

In the container, using the above environment variables/context, you can now analyze the dataset for the current period in your custom code. Once this analysis is completed, you can chose to emit your reports to be uploaded to S3. The reports that the pre-built container generates are documented in Container Contract Outputs (p. 633). If you would like the visualization of the reports to work in Amazon SageMaker Studio, you should follow the same format. You can also choose to emit completely custom reports.

You also have an option to emit CloudWatch metrics from the container by following the instructions in https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-byoc-cloudwatch.html.

 Topics
• Container Contract Inputs (p. 632)
• Container Contract Outputs (p. 633)
• CloudWatch Metrics (p. 638)

Container Contract Inputs

The Amazon SageMaker Model Monitor platform invokes your container code according to a specified schedule. If you chose to write your own container code, the following environment variables are available for your container code. In this context, you can analyze the current dataset or evaluate the constraints if you chose to and emit metrics, if applicable.

"Environment": {
"dataset_format": "\"sagemakerCaptureJson\": \"endpointInput\", \"endpointOutput\"\}"
"dataset_source": "/opt/ml/processing/endpointdata",
"end_time": "2019-12-01T16: 20: 00Z",
"output_path": "/opt/ml/processing/resultdata",
"publish_cloudwatch_metrics": "Disabled",
"sagemaker_endpoint_name": "endpoint-name",
"sagemaker_monitoring_schedule_name": "schedule-name",
"start_time": "2019-12-01T15: 20: 00Z"
}

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset_format</td>
<td>For a job started from a MonitoringSchedule backed by an Endpoint, this is sagemakerCaptureJson with the capture indices endpointInput, or endpointOutput, or both.</td>
</tr>
<tr>
<td>dataset_source</td>
<td>The local path in which the data corresponding to the monitoring period, as specified by start_time and end_time, are available. At this path, the data is available in /{endpoint-name}/{variant-name}/{yyyy/mm/dd}/hh. We sometimes download more than what is specified by the start and end times. It is up to the container code to parse the data as required.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>output_path</td>
<td>The local path to write output reports and other files. You specify this parameter in the CreateMonitoringSchedule request as MonitoringOutputConfig.MonitoringOutput[0].LocalPath. It is uploaded to the S3Uri path specified in MonitoringOutputConfig.MonitoringOutput[0].S3Uri.</td>
</tr>
<tr>
<td>publish_cloudwatch_metrics</td>
<td>For a job launched by CreateMonitoringSchedule, this parameter is set to Enabled. The container can choose to write the Amazon CloudWatch output file at [filepath].</td>
</tr>
<tr>
<td>sagemaker_endpoint_name</td>
<td>The name of the Endpoint that this scheduled job was launched for.</td>
</tr>
<tr>
<td>sagemaker_monitoring_schedule_name</td>
<td>The name of the MonitoringSchedule that launched this job.</td>
</tr>
<tr>
<td><em>sagemaker_endpoint_datacapture_prefix</em></td>
<td>The prefix specified in the DataCaptureConfig parameter of the Endpoint. The container can use this if it needs to directly access more data than already downloaded by Amazon SageMaker at the dataset_source path.</td>
</tr>
<tr>
<td>start_time, end_time</td>
<td>The time window for this analysis run. For example, for a job scheduled to run at 05:00 UTC and a job that runs on 20/02/2020, start_time: is 2020-02-19T06:00:00Z and end_time: is 2020-02-20T05:00:00Z.</td>
</tr>
<tr>
<td>baseline_constraints:</td>
<td>The local path of the baseline constraint file specified in BaselineConfig.ConstraintResource.S3Uri. This is available only if this parameter was specified in the CreateMonitoringSchedule request.</td>
</tr>
<tr>
<td>baseline_statistics</td>
<td>The local path to the baseline statistics file specified in BaselineConfig.StatisticsResource.S3Uri. This is available only if this parameter was specified in the CreateMonitoringSchedule request.</td>
</tr>
</tbody>
</table>

**Container Contract Outputs**

The container can analyze the data available in the *dataset_source* path and write reports to the path in *output_path*. The container code can write any reports that suit your needs.

If you use the following structure and contract, certain output files are treated specially by Amazon SageMaker in the visualization and API affordances This applies only to tabular datasets.
### Table: Output Files for Tabular Datasets

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statistics.json</td>
<td>This file is expected to have columnar statistics for each feature in the dataset that is analyzed. See the schema for this file in the next section.</td>
</tr>
<tr>
<td>constraints.json</td>
<td>This file is expected to have the constraints on the features observed. See the schema of this file below.</td>
</tr>
<tr>
<td>constraints_violations.json</td>
<td>This file is expected to have the list of violations found in this current set of data as compared to the baseline statistics and constraints file specified in the baseline_constraints and baseline_statistics path.</td>
</tr>
</tbody>
</table>

In addition, if the publish_cloudwatch_metrics value is "Enabled" container code can emit Amazon CloudWatch metrics in this location: /opt/ml/output/metrics/cloudwatch. The schema for these files is described in the following sections.

**Topics**

- Schema for Statistics (statistics.json file) (p. 634)
- Schema for Constraints (constraints.json file) (p. 636)

### Schema for Statistics (statistics.json file)

The schema defined in the statistics.json file specifies the statistical parameters to be calculated for the baseline and data that is captured. It also configures the bucket to be used by KLL, a very compact quantiles sketch with lazy compaction scheme.

```JSON
{
    "version": 0,
    # dataset level stats
    "dataset": {
        "item_count": number
    },
    # feature level stats
    "features": [
        {
            "name": "feature-name",
            "inferred_type": "Fractional" | "Integral",
            "numerical_statistics": {
                "common": {
                    "num_present": number,
                    "num_missing": number
                },
                "mean": number,
                "sum": number,
                "std_dev": number,
                "min": number,
                "max": number,
                "distribution": {
                    "kll": {
                        "buckets": [
                            {
                                "lower_bound": number,
                                "upper_bound": number,
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```
Note the following:

- The specified metrics are recognized by Amazon SageMaker in later visualization changes. The container can emit more metrics if required.

- **KLL sketch** is the recognized sketch. Custom containers can write their own representation, but it won’t be recognized by Amazon SageMaker in visualizations.

- By default, the distribution is materialized in 10 buckets. You can’t change this.
Schema for Constraints (constraints.json file)

A constraints.json file is used to express the constraints that a dataset must satisfy. Amazon SageMaker Model Monitor containers can use the constraints.json file to evaluate datasets against. Pre-built containers provide the ability to generate the constraints.json file automatically for a baseline dataset. If you bring your own container, you can provide it with similar abilities or you can create the constraints.json file in some other way. Here is the schema for the constraint file that the prebuilt container uses. Bring our own containers can adopt the same format or enhance it as required.

```json
{
    "version": 0,
    "features": [
        {
            "name": "string",
            "inferred_type": "Integral" | "Fractional" | "String" | "Unknown",
            "completeness": number, # denotes observed non-null value percentage
            "num_constraints": {
                "is_non_negative": boolean,
            },
            "string_constraints": {
                "domains": ["list of", "observed values", "for small cardinality"],
            },
            "monitoringConfigOverrides": {
                #monitoringConfigOverrides
            }
        }
    ]
}
```

# options to control monitoring for this feature with monitoring jobs
# See the following table for notes on what each constraint is doing.
"monitoring_config": {
    "evaluate_constraints": "Enabled",
    "emit_metrics": "Enabled",
    "datatype_check_threshold": 1.0,
    "domain_content_threshold": 1.0,
    "distribution_constraints": {
        "perform_comparison": "Enabled",
        "comparison_threshold": 0.1,
        "comparison_method": "Simple"||"Robust"
    }
}

Table: Monitoring Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluate_constraints</td>
<td>When Enabled, evaluates whether the current dataset being analysed satisfies the constraints specified in the constraints.json file taken as a baseline.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Enabled or Disabled</td>
</tr>
<tr>
<td></td>
<td>Default: Enabled</td>
</tr>
<tr>
<td>emit_metrics</td>
<td>When Enabled, emits CloudWatch metrics for the data contained in the file.</td>
</tr>
<tr>
<td>Constraint</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>Constraint</strong></td>
</tr>
<tr>
<td></td>
<td><strong>datatype_check_threshold</strong></td>
</tr>
<tr>
<td>Constraint</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>comparison_method</td>
<td>Whether to calculate linf_simple or linf_robust. The linf_simple is based on the maximum absolute difference between the cumulative distribution functions of two distributions. Calculating linf_robust is based on linf_simple, but is used when there are not enough samples. The linf_robust formula is based on the Two-sample Kolmogorov–Smirnov test. Valid values: linf_simple or linf_robust.</td>
</tr>
</tbody>
</table>

CloudWatch Metrics

If the publish_cloudwatch_metrics value is Enabled in the Environment map in the /opt/ml/processing/processingjobconfig.json file, the container code emits Amazon CloudWatch metrics in this location: /opt/ml/output/metrics/cloudwatch.

The schema for this file is closely based on the CloudWatch PutMetrics API. The namespace is not specified here. It defaults to /aws/sagemaker/Endpoint/data-metrics. However, you can specify dimensions. We recommend that you add the Endpoint and MonitoringSchedule dimensions at a minimum.

```json
{
   "MetricName": "", # Required
   "Timestamp": "2019-11-26T03:00:00Z", # Required
   "Dimensions": [{"Name": "Endpoint", "Value": "endpoint_0"},
    {"Name": "MonitoringSchedule", "Value": "schedule_0"}]
   "Value": Float,
   # Either the Value or the StatisticValues field can be populated and not both.
   "StatisticValues": {
      "SampleCount": Float,
      "Sum": Float,
      "Minimum": Float,
      "Maximum": Float
    },
   "Unit": "Count", # Optional
}
```

Create a Monitoring Schedule with an AWS CloudFormation Custom Resource

To use AWS CloudFormation to create a monitoring schedule, use an AWS CloudFormation custom resource. The custom resource is in Python. To deploy it, see Python Lambda deployment.

Custom Resource

Start by adding a custom resource to your AWS CloudFormation template. This will point to a AWS Lambda function that you create next.

This resource allows you to customize the parameters for the monitoring schedule. You can add or remove more parameters by modifying the AWS CloudFormation resource and the Lambda function in the following example resource.
Lambda Custom Resource Code

This AWS CloudFormation custom resource uses the Custom Resource Helper AWS library, which you can install with pip using pip install crhelper.

This Lambda function is invoked by AWS CloudFormation during the creation and deletion of the stack. This Lambda function is responsible for creating and deleting the monitoring schedule and using the parameters defined in the custom resource described in the preceding section.

```python
import boto3
import botocore
import logging
from crhelper import CfnResource
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
sm = boto3.client('sagemaker')

# cfhelper makes it easier to implement a CloudFormation custom resource
helper = CfnResource()

# CFN Handlers

def handler(event, context):
    helper(event, context)

@helper.create
def create_handler(event, context):
    """
    Called when CloudFormation custom resource sends the create event
    """
```
create_monitoring_schedule(event)

@helper.delete
def delete_handler(event, context):
    '''
    Called when CloudFormation custom resource sends the delete event
    '''
    schedule_name = get_schedule_name(event)
delete_monitoring_schedule(schedule_name)

@helper.poll_create
def poll_create(event, context):
    '''
    Return true if the resource has been created and false otherwise so
    CloudFormation polls again.
    '''
    schedule_name = get_schedule_name(event)
    logger.info('Polling for creation of schedule: %s', schedule_name)
    return is_schedule_ready(schedule_name)

@helper.update
def noop():
    '''
    Not currently implemented but crhelper will throw an error if it isn't added
    '''
    pass

# Helper Functions
def get_schedule_name(event):
    return event['ResourceProperties']['ScheduleName']

def create_monitoring_schedule(event):
    schedule_name = get_schedule_name(event)
    monitoring_schedule_config = create_monitoring_schedule_config(event)
    logger.info('Creating monitoring schedule with name: %s', schedule_name)
    sm.create_monitoring_schedule(
        MonitoringScheduleName=schedule_name,
        MonitoringScheduleConfig=monitoring_schedule_config)

def is_schedule_ready(schedule_name):
    is_ready = False
    schedule = sm.describe_monitoring_schedule(MonitoringScheduleName=schedule_name)
    status = schedule['MonitoringScheduleStatus']
    if status == 'Scheduled':
        logger.info('Monitoring schedule (%s) is ready', schedule_name)
        is_ready = True
    elif status == 'Pending':
        logger.info('Monitoring schedule (%s) still creating, waiting and polling
again...', schedule_name)
    else:
        raise Exception('Monitoring schedule ({}) has unexpected status:
{}',format(schedule_name, status))
    return is_ready

def create_monitoring_schedule_config(event):
    props = event['ResourceProperties']
    return {
"ScheduleConfig": {  
  "ScheduleExpression": props["ScheduleExpression"],  
},  
"MonitoringJobDefinition": {  
  "BaselineConfig": {  
    "ConstraintsResource": {  
      "S3Uri": props["BaselineConstraintsUri"],  
    },  
    "StatisticsResource": {  
      "S3Uri": props["BaselineStatisticsUri"],  
    }  
  },  
  "MonitoringInputs": [  
    {  
      "EndpointInput": {  
        "EndpointName": props["EndpointName"],  
        "LocalPath": props["InputLocalPath"],  
      }  
    }  
  ],  
  "MonitoringOutputConfig": {  
    "MonitoringOutputs": [  
      {  
        "S3Output": {  
          "S3Uri": props["OutputS3URI"],  
          "LocalPath": props["OutputLocalPath"],  
        }  
      }  
    ],  
  },  
  "MonitoringResources": {  
    "ClusterConfig": {  
      "InstanceCount": 1,  
      "InstanceType": "ml.t3.medium",  
      "VolumeSizeInGB": 50,  
    }  
  },  
  "MonitoringAppSpecification": {  
    "ImageUri": props["ImageURI"],  
    "RecordPreprocessorSourceUri": props["PostAnalyticsProcessorSourceUri"],  
    "PostAnalyticsProcessorSourceUri": props["PostAnalyticsProcessorSourceUri"],  
  },  
  "StoppingCondition": {  
    "MaxRuntimeInSeconds": 300  
  },  
  "RoleArn": props["PassRoleArn"],  
}  
}  
)

```
def delete_monitoring_schedule(schedule_name):
    logger.info('Deleting schedule: %s', schedule_name)
    try:
        sm.delete_monitoring_schedule(MonitoringScheduleName=schedule_name)
    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFound':
            logger.info('Resource not found, nothing to delete')
        else:
            logger.error('Unexpected error while trying to delete monitoring schedule')
            raise e
```
Deploy an Inference Pipeline

An *inference pipeline* is an Amazon SageMaker model that is composed of a linear sequence of two to five containers that process requests for inferences on data. You use an inference pipeline to define and deploy any combination of pretrained Amazon SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. You can use an inference pipeline to combine preprocessing, predictions, and post-processing data science tasks. Inference pipelines are fully managed.

You can add Amazon SageMaker Spark ML Serving and scikit-learn containers that reuse the data transformers developed for training models. The entire assembled inference pipeline can be considered as an Amazon SageMaker model that you can use to make either real-time predictions or to process batch transforms directly without any external preprocessing.

Within an inference pipeline model, Amazon SageMaker handles invocations as a sequence of HTTP requests. The first container in the pipeline handles the initial request, then the intermediate response is sent as a request to the second container, and so on, for each container in the pipeline. Amazon SageMaker returns the final response to the client.

When you deploy the pipeline model, Amazon SageMaker installs and runs all of the containers on each Amazon Elastic Compute Cloud (Amazon EC2) instance in the endpoint or transform job. Feature processing and inferences run with low latency because the containers are co-located on the same EC2 instances. You define the containers for a pipeline model using the `CreateModel` operation or from the console. Instead of setting one `PrimaryContainer`, you use the `Containers` parameter to set the containers that make up the pipeline. You also specify the order in which the containers are executed.

A pipeline model is immutable, but you can update an inference pipeline by deploying a new one using the `UpdateEndpoint` operation. This modularity supports greater flexibility during experimentation.

There are no additional costs for using this feature. You pay only for the instances running on an endpoint.

**Topics**

- Sample Notebooks for Inference Pipelines (p. 642)
- Feature Processing with Spark ML and Scikit-learn (p. 643)
- Create a Pipeline Model (p. 643)
- Run Real-time Predictions with an Inference Pipeline (p. 646)
- Run Batch Transforms with Inference Pipelines (p. 648)
- Inference Pipeline Logs and Metrics (p. 649)
- Troubleshoot Inference Pipelines (p. 654)

**Sample Notebooks for Inference Pipelines**

For a sample notebook that uploads and processes a dataset, trains a model, and builds a pipeline model, see the *Inference Pipelines with Spark ML and XGBoost on Abalone* notebook. This notebook shows how you can build your machine learning pipeline by using Spark feature Transformers and the Amazon SageMaker XGBoost algorithm. After training the model, the sample shows how to deploy the pipeline (feature Transformer and XGBoost) for real-time predictions and also performs a batch transform job using the same pipeline.

For more examples that show how to create and deploy inference pipelines, see the *Inference Pipelines with SparkML and BlazingText on DBPedia* and *Training using SparkML on EMR and hosting on*
SageMaker sample notebooks. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201).

To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the SageMaker Examples tab. There are three inference pipeline notebooks. The first two inference pipeline notebooks just described are located in the advanced_functionality folder and the third notebook is in the sagemaker-python-sdk folder. To open a notebook, choose its Use tab, then choose Create copy.

Feature Processing with Spark ML and Scikit-learn

Before training a model with either Amazon SageMaker built-in algorithms or custom algorithms, you can use Spark and scikit-learn preprocessors to transform your data and engineer features.

Feature Processing with Spark ML

You can run Spark ML jobs with AWS Glue, a serverless ETL (extract, transform, load) service, from your Amazon SageMaker notebook. You can also connect to existing EMR clusters to run Spark ML jobs with Amazon EMR. To do this, you need an AWS Identity and Access Management (IAM) role that grants permission for making calls from your Amazon SageMaker notebook to AWS Glue.

Note
To see which Python and Spark versions AWS Glue supports, refer to AWS Glue Release Notes.

After engineering features, you package and serialize Spark ML jobs with MLeap into MLeap containers that you can add to an inference pipeline. You don't need to use externally managed Spark clusters. With this approach, you can seamlessly scale from a sample of rows to terabytes of data. The same transformers work for both training and inference, so you don't need to duplicate preprocessing and feature engineering logic or develop a one-time solution to make the models persist. With inference pipelines, you don't need to maintain outside infrastructure, and you can make predictions directly from data inputs.

When you run a Spark ML job on AWS Glue, a Spark ML pipeline is serialized into MLeap format. Then, you can use the job with the SparkML Model Serving Container in an Amazon SageMaker Inference Pipeline. MLeap is a serialization format and execution engine for machine learning pipelines. It supports Spark, Scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

Feature Processing with Sci-kit Learn

You can run and package scikit-learn jobs into containers directly in Amazon SageMaker. For an example of Python code for building a scikit-learn featurizer model that trains on Fisher's Iris flower data set and predicts the species of Iris based on morphological measurements, see IRIS Training and Prediction with Sagemaker Scikit-learn.

Create a Pipeline Model

To create a pipeline model that can be deployed to an endpoint or used for a batch transform job, use the Amazon SageMaker console or the CreateModel operation.

To create an inference pipeline (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
2. Choose **Models**, and then choose **Create models** from the **Inference** group.

3. On the **Create model** page, provide a model name, choose an IAM role, and, if you want to use a private VPC, specify VPC values.

4. To add information about the containers in the inference pipeline, choose **Add container**, then choose **Next**.

5. Complete the fields for each container in the order that you want to execute them, up to the maximum of five. Complete the **Container input options**, **Location of inference code image**, and, optionally, **Location of model artifacts**, **Container host name**, and **Environmental variables** fields.
### Container definition 1

**Container input options**

- Provide model artifacts and inference image.

**Provide model artifacts and inference image**

- **Location of inference code image**
  The registry path where the inference code image is stored in Amazon ECR.
  
  123456789012.dkr.ecr.us-east-2.amazonaws.com/myimagerv1

- **Location of model artifacts - optional**
  The URL for the S3 location where model artifacts are stored.

  s3://bucket/path-to-your-data/

  The path must point to a single gzip-compressed tar archive (.tar.gz suffix).

- **Container host name - optional**
  The DNS host name for the container.

  Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

**Environment variables - optional**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>key1</td>
<td>value1</td>
<td></td>
</tr>
<tr>
<td>key2</td>
<td>value2</td>
<td></td>
</tr>
</tbody>
</table>

Add environment variable

### Container definition 2 - optional

**Container input options**

- Provide model artifacts and inference image.

**Provide model artifacts and inference image**

- **Location of inference code image**
  The registry path where the inference code image is stored in Amazon ECR.

  123456789012.dkr.ecr.us-east-2.amazonaws.com/myimagerv1

- **Location of model artifacts - optional**
  The URL for the S3 location where model artifacts are stored.

  s3://bucket/path-to-your-data/

  The path must point to a single gzip-compressed tar archive (.tar.gz suffix).

- **Container host name - optional**
  The DNS host name for the container.

  Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

**Environment variables - optional**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Remove</th>
</tr>
</thead>
</table>

Add environment variable

### Container definition 3 - optional

**Container input options**

- Provide model artifacts and inference image.

**Provide model artifacts and inference image**

- **Location of inference code image**
  The registry path where the inference code image is stored in Amazon ECR.

  123456789012.dkr.ecr.us-east-2.amazonaws.com/myimagerv1
The **MyInferencePipelineModel** page summarizes the settings for the containers that provide input for the model. If you provided the environment variables in a corresponding container definition, Amazon SageMaker shows them in the **Environment variables** field.

### Run Real-time Predictions with an Inference Pipeline

You can use trained models in an inference pipeline to make real-time predictions directly without performing external preprocessing. When you configure the pipeline, you can choose to use the built-in feature transformers already available in Amazon SageMaker. Or, you can implement your own transformation logic using just a few lines of scikit-learn or Spark code.

**MLeap**, a serialization format and execution engine for machine learning pipelines, supports Spark, scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

The containers in a pipeline listen on the port specified in the `SAGEMAKER_BIND_TO_PORT` environment variable (instead of 8080). When running in an inference pipeline, Amazon SageMaker automatically provides this environment variable to containers. If this environment variable isn’t present, containers...
default to using port 8080. To indicate that your container complies with this requirement, use the
following command to add a label to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

If your container needs to listen on a second port, choose a port in the range specified by the
SAGEMAKER_SAFE_PORT_RANGE environment variable. Specify the value as an inclusive range in the
format "xxxx-yyyy", where xxxx and yyyy are multi-digit integers. Amazon SageMaker provides this
value automatically when you run the container in a multicontainer pipeline.

**Note**
To use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms,
you need an Amazon Elastic Container Registry (Amazon ECR) policy. Your Amazon ECR
repository must grant Amazon SageMaker permission to pull the image. For more information,
see Troubleshoot Amazon ECR Permissions for Inference Pipelines (p. 654).

### Create and Deploy an Inference Pipeline Endpoint

The following code creates and deploys a real-time inference pipeline model with SparkML and XGBoost
models in series using the Amazon SageMaker SDK.

```python
from sagemaker.model import Model
from sagemaker.pipeline_model import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = s3://{}/{}/'.format(s3_model_bucket, s3_model_key_prefix, 'model.tar.gz')
sparkml_model = SparkMLModel(model_data=sparkml_data)
xgb_model = Model(model_data=xgb_model.model_data, image=training_image)

model_name = 'serial-inference-' + timestamp_prefix
endpoint_name = 'serial-inference-ep-' + timestamp_prefix
sm_model = PipelineModel(name=model_name, role=role, models=[sparkml_model, xgb_model])
sm_model.deploy(initial_instance_count=1, instance_type='ml.c4.xlarge',
endpoint_name=endpoint_name)
```

### Request Real-Time Inference from an Inference Pipeline Endpoint

The following example shows how to make real-time predictions by calling an inference endpoint and
passing a request payload in JSON format:

```python
from sagemaker.predictor import json_serializer, json_deserializer, RealTimePredictor
from sagemaker.content_types import CONTENT_TYPE_CSV, CONTENT_TYPE_JSON

payload = {
    "input": [
        {
            "name": "Pclass",
            "type": "float",
            "val": "1.0"
        },
        {
            "name": "Embarked",
            "type": "string",
            "val": "Q"
        },
        {
            "name": "Age",
            "type": "double",
```
Run Batch Transforms with Inference Pipelines

To get inferences on an entire dataset you run a batch transform on a trained model. To run inferences on a full dataset, you can use the same inference pipeline model created and deployed to an endpoint for real-time processing in a batch transform job. To run a batch transform job in a pipeline, you download the input data from Amazon S3 and send it in one or more HTTP requests to the inference pipeline model. For an example that shows how to prepare data for a batch transform, see the "Preparing Data for Batch Transform" section of the ML Pipeline with SparkML and XGBoost - Training and Inference sample notebook. For information about Amazon SageMaker batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 11).

Note
To use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon Elastic Container Registry (Amazon ECR) policy. Your Amazon ECR repository must grant Amazon SageMaker permission to pull the image. For more information, see Troubleshoot Amazon ECR Permissions for Inference Pipelines (p. 654).

The following example shows how to run a transform job using the Amazon SageMaker Python SDK. In this example, model_name is the inference pipeline that combines SparkML and XGBoost models (created in previous examples). The Amazon S3 location specified by input_data_path contains the input data, in CSV format, to be downloaded and sent to the Spark ML model. After the transform job has finished, the Amazon S3 location specified by output_data_path contains the output data returned by the XGBoost model in CSV format.

```python
import boto3

default_bucket = 'default_bucket'
default_key = 'key'
file_name = 'file_name'
CONTENT_TYPE_JSON = 'application/json'
CONTENT_TYPE_CSV = 'text/csv'

input_data_path = 's3://{}/{}/{}'.format(default_bucket, 'key', 'file_name')
output_data_path = 's3:///{}/{}'.format(default_bucket, 'key')
transform_job = sagemaker.transformer.Transformer(
    model_name = model_name,
    instance_count = 1,
    instance_type = 'ml.m4.xlarge',
    content_type=CONTENT_TYPE_JSON, accept=CONTENT_TYPE_CSV)

print(transform_job.predict(input_data_path))
```
strategy = 'SingleRecord',
assemble_with = 'Line',
output_path = output_data_path,
base_transform_job_name='inference-pipelines-batch',
sagemaker_session=sess,
accept = CONTENT_TYPE_CSV)
transform_job.transform(data = input_data_path,
content_type = CONTENT_TYPE_CSV,
split_type = 'Line')

Inference Pipeline Logs and Metrics

Monitoring is important for maintaining the reliability, availability, and performance of Amazon SageMaker resources. To monitor and troubleshoot inference pipeline performance, use Amazon CloudWatch logs and error messages. For information about the monitoring tools that Amazon SageMaker provides, see Monitor Amazon SageMaker (p. 712).

Use Metrics to Monitor Multi-container Models

To monitor the multi-container models in Inference Pipelines, use Amazon CloudWatch. CloudWatch collects raw data and processes it into readable, near real-time metrics. Amazon SageMaker training jobs and endpoints write CloudWatch metrics and logs in the AWS/SageMaker namespace.

The following tables list the metrics and dimensions for the following:

- Endpoint invocations
- Training jobs, batch transform jobs, and endpoint instances

A *dimension* is a name/value pair that uniquely identifies a metric. You can assign up to 10 dimensions to a metric. For more information on monitoring with CloudWatch, see Monitor Amazon SageMaker with Amazon CloudWatch (p. 712).

Endpoint Invocation Metrics

The AWS/SageMaker namespace includes the following request metrics from calls to `InvokeEndpoint` (p. 1260).

Metrics are reported at a 1-minute intervals.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation4XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests that the model returned a 4xx HTTP response code for. For each 4xx response, Amazon SageMaker sends a 1.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocation5XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests that the model returned a 5xx HTTP response code for. For each 5xx response, Amazon SageMaker sends a 1.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
</tbody>
</table>
## Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invocations</strong></td>
<td>The number of <code>InvokeEndpoint</code> requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the <code>Sum</code> statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <code>Sum</code>, <code>Sample Count</code></td>
</tr>
<tr>
<td><strong>InvocationsPerInstance</strong></td>
<td>The number of endpoint invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code>. Amazon SageMaker sends ( \frac{1}{\text{numberOfInstances}} ) as the value for each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> at the endpoint at the time of the request.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <code>Sum</code></td>
</tr>
<tr>
<td><strong>ModelLatency</strong></td>
<td>The time the model or models took to respond. This includes the time it took to send the request, to fetch the response from the model container, and to complete the inference in the container. <code>ModelLatency</code> is the total time taken by all containers in an inference pipeline.</td>
</tr>
<tr>
<td></td>
<td>Units: Microseconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></td>
</tr>
<tr>
<td><strong>OverheadLatency</strong></td>
<td>The time added to the time taken to respond to a client request by Amazon SageMaker for overhead. <code>OverheadLatency</code> is measured from the time that Amazon SageMaker receives the request until it returns a response to the client, minus the <code>ModelLatency</code>. Overhead latency can vary depending on request and response payload sizes, request frequency, and authentication or authorization of the request, among other factors.</td>
</tr>
<tr>
<td></td>
<td>Units: Microseconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></td>
</tr>
<tr>
<td><strong>ContainerLatency</strong></td>
<td>The time it took for an Inference Pipelines container to respond as viewed from Amazon SageMaker. <code>ContainerLatency</code> includes the time it took to send the request, to fetch the response from the model's container, and to complete inference in the container.</td>
</tr>
<tr>
<td></td>
<td>Units: Microseconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></td>
</tr>
</tbody>
</table>

### Dimensions for Endpoint Invocation Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EndpointName</strong>, <strong>VariantName</strong>, <strong>ContainerName</strong></td>
<td>Filters endpoint invocation metrics for a <code>ProductionVariant</code> at the specified endpoint and for the specified variant.</td>
</tr>
</tbody>
</table>
For an inference pipeline endpoint, CloudWatch lists per-container latency metrics in your account as **Endpoint Container Metrics** and **Endpoint Variant Metrics** in the **SageMaker** namespace, as follows. The **ContainerLatency** metric appears only for inferences pipelines.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUtilization</td>
<td>The percentage of CPU units that are used by the containers running on an instance. The value ranges from 0% to 100%, and is multiplied by the number of CPUs. For example, if there are four CPUs, CPUUtilization can range from 0% to 400%. For training jobs, CPUUtilization is the CPU utilization of the algorithm container running on the instance. For batch transform jobs, CPUUtilization is the CPU utilization of the transform container running on the instance. For multi-container models, CPUUtilization is the sum of CPU utilization by all containers running on the instance. For endpoint variants, CPUUtilization is the sum of CPU utilization by all of the containers running on the instance. Units: Percent</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory that is used by the containers running on an instance. This value ranges from 0% to 100%. For training jobs, MemoryUtilization is the memory used by the algorithm container running on the instance. For batch transform jobs, MemoryUtilization is the memory used by the transform container running on the instance.</td>
</tr>
</tbody>
</table>

For each endpoint and each container, latency metrics display names for the container, endpoint, variant, and metric.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemoryUtilization</td>
<td>For multi-container models, <strong>MemoryUtilization</strong> is the sum of memory used by all containers running on the instance. For endpoint variants, <strong>MemoryUtilization</strong> is the sum of memory used by all of the containers running on the instance. Units: Percent</td>
</tr>
<tr>
<td>GPUUtilization</td>
<td>The percentage of GPU units that are used by the containers running on an instance. <strong>GPUUtilization</strong> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <strong>GPUUtilization</strong> can range from 0% to 400%. For training jobs, <strong>GPUUtilization</strong> is the GPU used by the algorithm container running on the instance. For batch transform jobs, <strong>GPUUtilization</strong> is the GPU used by the transform container running on the instance. For multi-container models, <strong>GPUUtilization</strong> is the sum of GPU used by all containers running on the instance. For endpoint variants, <strong>GPUUtilization</strong> is the sum of GPU used by all of the containers running on the instance. Units: Percent</td>
</tr>
<tr>
<td>GPUMemoryUtilization</td>
<td>The percentage of GPU memory used by the containers running on an instance. <strong>GPUMemoryUtilization</strong> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <strong>GPUMemoryUtilization</strong> can range from 0% to 400%. For training jobs, <strong>GPUMemoryUtilization</strong> is the GPU memory used by the algorithm container running on the instance. For batch transform jobs, <strong>GPUMemoryUtilization</strong> is the GPU memory used by the transform container running on the instance. For multi-container models, <strong>GPUMemoryUtilization</strong> is sum of GPU used by all containers running on the instance. For endpoint variants, <strong>GPUMemoryUtilization</strong> is the sum of the GPU memory used by all of the containers running on the instance. Units: Percent</td>
</tr>
<tr>
<td>DiskUtilization</td>
<td>The percentage of disk space used by the containers running on an instance. <strong>DiskUtilization</strong> ranges from 0% to 100%. This metric is not supported for batch transform jobs. For training jobs, <strong>DiskUtilization</strong> is the disk space used by the algorithm container running on the instance. For endpoint variants, <strong>DiskUtilization</strong> is the sum of the disk space used by all of the provided containers running on the instance. Units: Percent</td>
</tr>
</tbody>
</table>
Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>For training jobs, Host has the format <code>[training-job-name]/algo-[instance-number-in-cluster]</code>. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the <code>/aws/sagemaker/TrainingJobs</code> namespace. For batch transform jobs, Host has the format <code>[transform-job-name]/[instance-id]</code>. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the <code>/aws/sagemaker/TransformJobs</code> namespace. For endpoints, Host has the format <code>[endpoint-name]/[production-variant-name]/[instance-id]</code>. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the <code>/aws/sagemaker/Endpoints</code> namespace.</td>
</tr>
</tbody>
</table>

To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, Amazon SageMaker also sends anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` to Amazon CloudWatch Logs. You can use this information for debugging and to analyze progress.

Use Logs to Monitor an Inference Pipeline

The following table lists the log groups and log streams Amazon SageMaker sends to Amazon CloudWatch.

A log stream is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A log group is a group of log streams that share the same retention, monitoring, and access control settings.

**Logs**

<table>
<thead>
<tr>
<th>Log Group Name</th>
<th>Log Stream Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/aws/sagemaker/TrainingJobs</code></td>
<td><code>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</code></td>
</tr>
<tr>
<td><code>/aws/sagemaker/Endpoints/[EndpointName]</code></td>
<td><code>[production-variant-name]/[instance-id]</code></td>
</tr>
<tr>
<td></td>
<td><code>[production-variant-name]/[instance-id]/[container-name provided in the Amazon SageMaker model]</code> (For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses <strong>container-1, container-2</strong>, and so on, in the order that containers are provided in the model.</td>
</tr>
<tr>
<td><code>/aws/sagemaker/NotebookInstances</code></td>
<td><code>[notebook-instance-name]/[LifecycleConfigHook]</code></td>
</tr>
<tr>
<td><code>/aws/sagemaker/TransformJobs</code></td>
<td><code>[transform-job-name]/[instance-id]-[epoch_timestamp]</code></td>
</tr>
<tr>
<td></td>
<td><code>[transform-job-name]/[instance-id]-[epoch_timestamp]/data-log</code></td>
</tr>
</tbody>
</table>
Log Group Name | Log Stream Name
---------------|-------------------
```plaintext
[transform-job-name]/[instance-id]-[epoch_timestamp]/
[container-name provided in the Amazon SageMaker model]
(For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.
```

**Note**
Amazon SageMaker creates the `/aws/sagemaker/NotebookInstances` log group when you create a notebook instance with a lifecycle configuration. For more information, see Customize a Notebook Instance (p. 206).

For more information about Amazon SageMaker logging, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 719).

**Troubleshoot Inference Pipelines**

To troubleshoot inference pipeline issues, use CloudWatch logs and error messages. If you are using custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you might also encounter permissions problems. To grant the required permissions, create an Amazon Elastic Container Registry (Amazon ECR) policy.

**Topics**
- Troubleshoot Amazon ECR Permissions for Inference Pipelines (p. 654)
- Use CloudWatch Logs to Troubleshoot Amazon SageMaker Inference Pipelines (p. 655)
- Use Error Messages to Troubleshoot Inference Pipelines (p. 655)

**Troubleshoot Amazon ECR Permissions for Inference Pipelines**

When you use custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon ECR policy. The policy allows your Amazon ECR repository to grant permission for Amazon SageMaker to pull the image. The policy must add the following permissions:

```json
{
   "Version": "2008-10-17",
   "Statement": [
      {
         "Sid": "allowSageMakerToPull",
         "Effect": "Allow",
         "Principal": {
            "Service": "sagemaker.amazonaws.com"
         },
         "Action": [
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability"
         ]
      }
   ]
}
```
Use CloudWatch Logs to Troubleshoot Amazon SageMaker Inference Pipelines

Amazon SageMaker publishes the container logs for endpoints that deploy an inference pipeline to Amazon CloudWatch at the following path for each container.

/path/sagemaker/Endpoints/{EndpointName}/{Variant}/{InstanceId}/{ContainerHostname}

For example, logs for this endpoint are published to the following log groups and streams:

<table>
<thead>
<tr>
<th>EndpointName</th>
<th>MyInferencePipelinesEndpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant</td>
<td>MyInferencePipelinesVariant</td>
</tr>
<tr>
<td>InstanceId</td>
<td>i-0179208609ff7e488</td>
</tr>
<tr>
<td>ContainerHostname</td>
<td>MyContainerName1 and MyContainerName2</td>
</tr>
</tbody>
</table>

\[logGroup: /aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint\]
\[logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName1\]
\[logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName2\]

A log stream is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A log group is a group of log streams that share the same retention, monitoring, and access control settings.

To see the log groups and streams

2. In the navigation page, choose Logs.
3. In Log Groups, filter on MyInferencePipelinesEndpoint:

4. To see the log streams, on the CloudWatch Log Groups page, choose MyInferencePipelinesEndpoint, and then Search Log Group.

For a list of the logs that Amazon SageMaker publishes, see Inference Pipeline Logs and Metrics (p. 649).

Use Error Messages to Troubleshoot Inference Pipelines

The inference pipeline error messages indicate which containers failed.
If an error occurs while Amazon SageMaker is invoking an endpoint, the service returns a `ModelError` (error code 424), which indicates which container failed. If the request payload (the response from the previous container) exceeds the limit of 5 MB, Amazon SageMaker provides a detailed error message, such as:

Received response from MyContainerName1 with status code 200. However, the request payload from MyContainerName1 to MyContainerName2 is 6000000 bytes, which has exceeded the maximum limit of 5 MB. See https://us-west-2.console.aws.amazon.com/cloudwatch/home?region=us-west-2#logEventViewer:group=/aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint in account 123456789012 for more information.

If a container fails the ping health check while Amazon SageMaker is creating an endpoint, it returns a `ClientError` and indicates all of the containers that failed the ping check in the last health check.

Use Batch Transform

Use batch transform when you need to do the following:

- Preprocess datasets to remove noise or bias that interferes with training or inference from your dataset.
- Get inferences from large datasets.
- Run inference when you don't need a persistent endpoint.
- Associate input records with inferences to assist the interpretation of results.

To filter input data before performing inferences or to associate input records with inferences about those records, see Associate Prediction Results with Input Records (p. 658). For example, you can filter input data to provide context for creating and interpreting reports about the output data.

For more information about batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 11).

Topics

- Use Batch Transform to Get Inferences from Large Datasets (p. 656)
- Speed up a Batch Transform Job (p. 658)
- Use Batch Transform to Test Production Variants (p. 658)
- Batch Transform Errors (p. 658)
- Batch Transform Sample Notebooks (p. 658)
- Associate Prediction Results with Input Records (p. 658)

Use Batch Transform to Get Inferences from Large Datasets

Batch transform automatically manages the processing of large datasets within the limits of specified parameters. For example, suppose that you have a dataset file, `input1.csv`, stored in an S3 bucket. The content of the input file might look like the following:

```
Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-AttributeM
```
When a batch transform job starts, Amazon SageMaker initializes compute instances and distributes the inference or preprocessing workload between them. When you have multiples files, one instance might process `input1.csv`, and another instance might process the file named `input2.csv`.

To keep large payloads below the `MaxPayloadInMB` limit, you can split an input file into several mini-batches. For example, you might create a mini-batch from `input1.csv` by including only two of the records.

Note
Amazon SageMaker processes each input file separately. It doesn't combine mini-batches from different input files to comply with the `MaxPayloadInMB` limit.

To split input files into mini-batches, when you create a batch transform job, set the `SplitType` parameter value to `Line`. If `SplitType` is set to `None` or if an input file can't be split into mini-batches, Amazon SageMaker uses the entire input file in a single request.

If the batch transform job successfully processes all of the records in an input file, it creates an output file with the same name and the `.out` file extension. For multiple input files, such as `input1.csv` and `input2.csv`, the output files are named `input1.csv.out` and `input2.csv.out`. The batch transform job stores the output files in the specified location in Amazon S3, such as `s3://awsexamplebucket/output/`.

The predictions in an output file are listed in the same order as the corresponding records in the input file. The output file `input1.csv.out`, based on the input file shown earlier, would look like the following.

To combine the results of multiple output files into a single output file, set the `AssembleWith` parameter to `Line`.

When the input data is very large and is transmitted using HTTP chunked encoding, to stream the data to the algorithm, set `MaxPayloadInMB` to 0. Amazon SageMaker built-in algorithms don't support this feature.

For information about using the API to create a batch transform job, see the `CreateTransformJob (p. 939)` API. For more information about the correlation between batch transform input and output objects, see `OutputDataConfig`. For an example of how to use batch transform, see `Step 6.2: Deploy the Model with Batch Transform (p. 37)`.
Speed up a Batch Transform Job

If you are using the CreateTransformJob API, you can reduce the time it takes to complete batch transform jobs by using optimal values for parameters such as MaxPayloadInMB, MaxConcurrentTransforms, or BatchStrategy. If you are using the Amazon SageMaker console, you can specify these optimal parameter values in the Additional configuration section of the Batch transform job configuration page. Amazon SageMaker automatically finds the optimal parameter settings for built-in algorithms. For custom algorithms, provide these values through an execution-parameters endpoint.

Use Batch Transform to Test Production Variants

To test different models or various hyperparameter settings, create a separate transform job for each new model variant and use a validation dataset. For each transform job, specify a unique model name and location in Amazon S3 for the output file. To analyze the results, use Inference Pipeline Logs and Metrics (p. 649).

Batch Transform Errors

Amazon SageMaker uses the Amazon S3 Multipart Upload API to upload results from a batch transform job to Amazon S3. If an error occurs, the uploaded results are removed from Amazon S3. In some cases, such as when a network outage occurs, an incomplete multipart upload might remain in Amazon S3. To avoid incurring storage charges, we recommend that you add the S3 bucket policy to the S3 bucket lifecycle rules. This policy deletes incomplete multipart uploads that might be stored in the S3 bucket. For more information, see Object Lifecycle Management.

If a batch transform job fails to process an input file because of a problem with the dataset, Amazon SageMaker marks the job as failed. If an input file contains a bad record, the transform job doesn't create an output file for that input file because doing so prevents it from maintaining the same order in the transformed data as in the input file. When your dataset has multiple input files, a transform job continues to process input files even if it fails to process one. The processed files still generate usable results.

Exceeding the MaxPayloadInMB limit causes an error. This might happen with a large dataset if it can't be split, the SplitType parameter is set to none, or individual records within the dataset exceed the limit.

If you are using your own algorithms, you can use placeholder text, such as ERROR, when the algorithm finds a bad record in an input file. For example, if the last record in a dataset is bad, the algorithm places the placeholder text for that record in the output file.

Batch Transform Sample Notebooks

For a sample notebook that uses batch transform with a principal component analysis (PCA) model to reduce data in a user-item review matrix, followed by the application of a density-based spatial clustering of applications with noise (DBSCAN) algorithm to cluster movies, see Batch Transform with PCA and DBSCAN Movie Clusters. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). After creating and opening a notebook instance, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the NTM algorithms are located in the Advanced functionality section. To open a notebook, choose its Use tab, then choose Create copy.

Associate Prediction Results with Input Records

When making predictions on a large dataset, you can exclude attributes that aren't needed for prediction. After the predictions have been made, you can associate some of the excluded attributes with
those predictions or with other input data in your report. By using batch transform to perform these data processing steps, you can often eliminate additional preprocessing or postprocessing. You can use input files in JSON and CSV format only.

**Topics**
- Workflow for Associating Inferences with Input Records (p. 659)
- Use Data Processing in Batch Transform Jobs (p. 659)
- Supported JSONPath Operators (p. 660)
- Batch Transform Examples (p. 661)

**Workflow for Associating Inferences with Input Records**

The following diagram shows the workflow for associating inferences with input records.

To associate inferences with input data, there are three main steps:

1. Filter the input data that is not needed for inference before passing the input data to the batch transform job. Use the `InputFilter` parameter to determine which attributes to use as input for the model.

2. Associate the input data with the inference results. Use the `JoinSource` parameter to combine the input data with the inference.

3. Filter the joined data to retain the inputs that are needed to provide context for interpreting the predictions in the reports. Use `OutputFilter` to store the specified portion of the joined dataset in the output file.

**Use Data Processing in Batch Transform Jobs**

When creating a batch transform job with `CreateTransformJob` to process data:
1. Specify the portion of the input to pass to the model with the InputFilter parameter in the DataProcessing data structure.

2. Join the raw input data with the transformed data with the JoinSource parameter.

3. Specify which portion of the joined input and transformed data from the batch transform job to include in the output file with the OutputFilter parameter.

4. Choose either JSON- or CSV-formatted files for input:
   - For JSON- or JSON Lines-formatted input files, Amazon SageMaker either adds the SageMakerOutput attribute to the input file or creates a new JSON output file with the SageMakerInput and SageMakerOutput attributes. For more information, see DataProcessing (p. 1328).
   - For CSV-formatted input files, the joined input data is followed by the transformed data and the output is a CSV file.

If you use an algorithm with the DataProcessing structure, it must support your chosen format for both input and output files. For example, with the TransformOutput (p. 1542) field of the CreateTransformJob API, you must set both the ContentType and Accept parameters to one of the following values: text/csv, application/json, or application/jsonlines. The syntax for specifying columns in a CSV file and specifying attributes in a JSON file are different. Using the wrong syntax causes an error. For more information, see Batch Transform Examples (p. 661). For more information about input and output file formats for built-in algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 220).

The record delimiters for the input and output must also be consistent with your chosen file input. The SplitType parameter indicates how to split the records in the input dataset. The AssemblyType parameter indicates how to reassemble the records for the output. If you set input and output formats to text/csv, you must also set the SplitType and AssemblyType parameters to line. If you set the input and output formats to application/jsonlines, you can set both SplitType and AssemblyType to either none or line.

For JSON files, the attribute name SageMakerOutput is reserved for output. The JSON input file can't have an attribute with this name. If it does, the data in the input file might be overwritten.

### Supported JSONPath Operators

To filter and join the input data and inference, use a JSONPath subexpression. The following table lists the supported JSONPath operators.

<table>
<thead>
<tr>
<th>JSONPath Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>The root element to a query. This operator is required at the beginning of all path expressions.</td>
<td>$</td>
</tr>
<tr>
<td>.&lt;name&gt;</td>
<td>A dot-notated child element.</td>
<td>$.id</td>
</tr>
<tr>
<td>*</td>
<td>A wildcard. Use in place of an attribute name or numeric value.</td>
<td>$.id.*</td>
</tr>
<tr>
<td>[ '&lt;name&gt;' (',' '&lt;name&gt;']</td>
<td>A bracket-notated element or multiple child elements.</td>
<td>['$[id]', 'SageMakerOutput']</td>
</tr>
<tr>
<td>[&lt;number&gt; (','&lt;number&gt;)]</td>
<td>An index or array of indexes. Negative index values are also supported. A -1 index refers to the last element in an array.</td>
<td>$[1], $[1, 3, 5]</td>
</tr>
</tbody>
</table>
Amazon SageMaker Developer Guide
Associate Prediction Results with Input

JSONPath Operator

Description

Example

[<start>:<end>]

An array slice operator. The array slice()
method extracts a section of an array and
returns a new array. If you omit <start>,
Amazon SageMaker uses the ﬁrst element
of the array. If you omit <end>, Amazon
SageMaker uses the last element of the array.

$[2:5], $[:5], $[2:]

Note

Amazon SageMaker supports only a subset of the deﬁned JSONPath operators. For more
information about JSONPath operators, see JsonPath on GitHub.

Batch Transform Examples
The following examples show some common ways to join input data with prediction results.
Topics
• Example: Output Only Inferences (p. 661)
• Example: Output Input Data and Inferences (p. 661)
• Example: Output an ID Column with Results and Exclude the ID Column from the Input
(CSV) (p. 662)
• Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input
(JSON) (p. 663)

Example: Output Only Inferences
By default, the DataProcessing parameter doesn't join inference results with input. It outputs only the
inference results.
If you want to explicitly specify to not join results with input, use the Amazon SageMaker Python SDK
and specify the following settings in a transformer call.
sm_transformer = sagemaker.transformer.Transformer(…)
sm_transformer.transform(…, input_filter="$", join_source= "None", output_filter="$")

To output inferences using the AWS SDK for Python, add the following code to your CreateTransformJob
request. The following code mimics the default behavior.
{

}

"DataProcessing": {
"InputFilter": "$",
"JoinSource": "None",
"OutputFilter": "$"
}

Example: Output Input Data and Inferences
If you're using the Amazon SageMaker Python SDK, to combine the input data with the inferences in the
output ﬁle, specify "Input" for the JoinSource parameter in a transformer call.
sm_transformer = sagemaker.transformer.Transformer(…)

661


sm_transformer.transform(_, join_source= "Input")

If you're using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your CreateTransformJob (p. 939) request.

```json
{
    "DataProcessing": {
        "JoinSource": "Input"
    }
}
```

For JSON or JSON Lines input files, the results are in the SageMakerOutput key in the input JSON file. For example, if the input is a JSON file that contains the key-value pair ("key":1), the data transform result might be {"label":1}.

Amazon SageMaker stores both in the input file in the SageMakerInput key.

```json
{
    "key":1,
    "SageMakerOutput": {"label":1}
}
```

**Note**
The joined result for JSON must be a key-value pair object. If the input isn't a key-value pair object, Amazon SageMaker creates a new JSON file. In the new JSON file, the input data is stored in the SageMakerInput key and the results are stored as the SageMakerOutput value.

For a CSV file, for example, if the record is [1, 2, 3] and the label result is [1], then the output file would contain [1, 2, 3, 1].

**Example: Output an ID Column with Results and Exclude the ID Column from the Input (CSV)**

If you are using the Amazon SageMaker Python SDK, to include results or an ID column in the output, specify indexes of the joined dataset in a transformer call. For example, if your data includes five columns and the first one is the ID column, use the following transformer request.

```python
sm_transformer = sagemaker.transformer.Transformer(…)
sm_transformer.transform(_, input_filter="$[1:]", join_source= "Input", output_filter="")
```

If you are using the AWS SDK for Python (Boto 3), add the following code to your CreateTransformJob (p. 939) request.

```json
{
    "DataProcessing": {
        "InputFilter": "$[1:]",
        "JoinSource": "Input",
        "OutputFilter": ""
    }
}
```

To specify columns in Amazon SageMaker, use the index of the array elements. The first column is index 0, the second column is index 1, and the sixth column is index 5.

To exclude the first column from the input, set InputFilter to "$[1:]". The colon (:) tells Amazon SageMaker to include all of the elements between two values, inclusive. For example, "$[1:4]" specifies the second through fifth columns.
If you omit the number after the colon, for example, `[5:]`, the subset includes all columns from the 6th column through the last column. If you omit the number before the colon, for example, `[5:]`, the subset includes all columns from the first column (index 0) through the sixth column.

**Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input (JSON)**

If you are using the Amazon SageMaker Python SDK, include results of an ID attribute in the output by specifying it in a transformer call. For example, if you store data in the `features` attribute and the record ID in the `ID` attribute, you would use the following transformer request.

```python
sm_transformer = sagemaker.transformer.Transformer(_)
sm_transformer.transform(_, input_filter="$.features", join_source= "Input",
output_filter="$['id','SageMakerOutput']")
```

If you are using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your `CreateTransformJob (p. 939)` request.

```json
{
   "DataProcessing": {
      "InputFilter": "$.features",
      "JoinSource": "Input",
      "OutputFilter": "$['id','SageMakerOutput']"
   }
}
```

**Warning**

If you are using a JSON-formatted input file, the file can't contain the attribute name `SageMakerOutput`. This attribute name is reserved for the output file. If your JSON-formatted input file contains an attribute with this name, values in the input file might be overwritten with the inference.

---

**Compile and Deploy Models with Amazon SageMaker Neo**

Neo is a new capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

Ordinarily, optimizing machine learning models for inference on multiple platforms is extremely difficult because you need to hand-tune models for the specific hardware and software configuration of each platform. If you want to get optimal performance for a given workload, you need to know the hardware architecture, instruction set, memory access patterns, and input data shapes among other factors. For traditional software development, tools such as compilers and profilers simplify the process. For machine learning, most tools are specific to the framework or to the hardware. This forces you into a manual trial-and-error process that is unreliable and unproductive.

Neo eliminates the time and effort required to do this by automatically optimizing TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models for deployment on ARM, Intel, and Nvidia processors. Neo consists of a compiler and a runtime. First, the Neo compilation API reads models exported from various frameworks. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimizations. Then it generates binary code for the optimized operations, writes them to a shared object library, and saves the model definition and parameters into separate files. Neo also provides a runtime for each target platform that loads and executes the compiled model.
You can create a Neo compilation job from either the Amazon SageMaker console, AWS Command Line Interface (AWS CLI), Python notebook, or the Amazon SageMaker SDK. With a few CLI commands, an API invocation, or a few clicks, you can convert a model for your chosen platform. You can deploy the model to an Amazon SageMaker endpoint or on an AWS IoT Greengrass device quickly. Amazon SageMaker provides Neo container images for Amazon SageMaker XGBoost and Image Classification models, and supports Amazon SageMaker-compatible containers for your own compiled models.

**Note**
Neo currently supports image classification models exported as frozen graphs from TensorFlow, MXNet, or PyTorch, and XGBoost models. Neo is available in the following AWS Regions where Amazon SageMaker is supported:

- **Asia Pacific** (Hong Kong, Mumbai, Seoul, Singapore, Sydney, Tokyo)
- **Canada** (Central)
- **EU** (Frankfurt, Ireland, London, Paris, Stockholm)
- **North America** (N. Virginia, Ohio, Oregon, N. California)
- **South America** (Sao Paulo)

**Topics**
- Amazon SageMaker Neo Sample Notebooks (p. 664)
- Use Neo to Compile a Model (p. 664)
- Deploy a Model (p. 669)
- Request Inferences from a Deployed Service (p. 677)
- Troubleshooting Neo Compilation Errors (p. 677)

**Amazon SageMaker Neo Sample Notebooks**

For sample notebooks that uses Amazon SageMaker Neo to train, compile, optimize, and deploy machine learning models to make inferences, see:

- MNIST Training, Compilation and Deployment with MXNet Module
- MNIST Training, Compilation and Deployment with Tensorflow Module
- Deploying pre-trained PyTorch vision models with Amazon SageMaker Neo
- Model Optimization with an Image Classification Example
- Model Optimization with XGBoost Example

For instructions on how to run these example notebooks in Amazon SageMaker, see Use Example Notebooks (p. 208). If you need instructions on how to create a notebook instance to run these examples, see Amazon SageMaker, see Use Amazon SageMaker Notebook Instances (p. 201). To navigate to the relevant example in your notebook instance, choose the Amazon SageMaker Examples tab to see a list of all of the Amazon SageMaker samples. To open a notebook, choose its Use tab, then choose Create copy.

**Use Neo to Compile a Model**

This section show how to create, describe, stop, and list compilation jobs. There are three options available in Neo for managing the compilation jobs for machine learning models: Using the Neo CLI, the Amazon SageMaker console, or the Amazon SageMaker SDK.

**Topics**

664
Compile Models

- Compile a Model (API) (p. 665)
- Compile a Model (Console) (p. 666)
- Compile a Model (Amazon SageMaker SDK) (p. 668)

Compile Models

This section shows how to manage compilation jobs for machine learning models. You can create, describe, stop, and list compilation jobs.

Create a Compilation Job

As shown in the following JSON file, you specify the data input format, the S3 bucket where you stored your model, the S3 bucket where you want to write the compiled model, and the target hardware:

```
job.json
{
    "CompilationJobName": "job002",
    "RoleArn": "arn:aws:iam::<your-account>:role/service-role/AmazonSageMaker-ExecutionRole-20180829T140091",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/train",
        "DataInputConfig": "{\"data\": [1,3,1024,1024]}",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/compile",
        "TargetDevice": "ml_c5"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}
```

```
aws sagemaker create-compilation-job \
--cli-input-json file://job.json \
--region us-west-2

# You should get CompilationJobArn
```

Describe a Compilation Job

```
aws sagemaker describe-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2
```

Stop a Compilation Job

```
aws sagemaker stop-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2

# There is no output for compilation-job operation
```

List a Compilation Job

```
aws sagemaker list-compilation-jobs \
```
Compile Models

Compile a Model (Console)

You can create a Neo compilation job in the Amazon SageMaker console. In the Amazon SageMaker console, choose Compilation jobs, and then choose Create compilation job.

On the Create compilation job page, for Job name, enter a name. Then select an IAM role.

If you don’t have an IAM role, choose Create a new role.

On the Create an IAM role page, choose Any S3 bucket, and choose Create role.
In the **Input configuration** section, for **Location of model artifacts**, enter the path of the S3 bucket that contains your model artifacts. For **Data input configuration**, enter the JSON string that specifies how many data matrix inputs you and the shape of each data matrices. For **Machine learning framework**, choose the framework.

In the **Output configuration** section, for **S3 Output location**, enter the path to the S3 bucket or folder where you want to store the model. For **Target device**, choose which device you want to deploy your model to, and choose **Create job**.
Compile a Model (Amazon SageMaker SDK)

Follow the steps described in the Running the Training Job section of the MNIST Training, Compilation and Deployment with MXNet Module sample to produce a machine learning model train using Amazon SageMaker. Then you can use Neo to further optimize the model with the following code:

```python
output_path = '/'.join( mnist_estimator.output_path.split('/')[:-1])
compiled_model = mnist_estimator.compile_model(target_instance_family='ml_c5',
                                              input_shape={'data':[1, 784]},
                                              role=role,
                                              output_path=output_path)
```
Deploy Models

This code compiles the model and saves the optimized model in output_path. Sample notebooks of using SDK are provided in the Amazon SageMaker Neo Sample Notebooks (p. 664) section.

Deploy a Model

You can deploy the compact module to performance-critical cloud services with Amazon SageMaker Hosting Services or to resource-constrained edge devices with AWS IoT Greengrass.

Topics

- Deploy a Model Compiled with Neo with Hosting Services (p. 669)
- Deploy a Model Compiled with Neo (AWS IoT Greengrass) (p. 676)

Deploy a Model Compiled with Neo with Hosting Services

To deploy a Neo-compiled model to an HTTPS endpoint, you must configure and create the endpoint for the model using Amazon SageMaker hosting services. Currently developers can use Amazon SageMaker APIs to deploy modules on to ml.c5, ml.c4, ml.m5, ml.m4, ml.p3, and ml.p2 instances.

When you deploy a compiled model, you need to use the same instance for the target that you used for compilation. This creates an Amazon SageMaker endpoint that you can use to perform inferences. There are three options available for deploying Neo-compiled models:

Topics

- Deploy a Model Compiled with Neo (AWS CLI) (p. 669)
- Deploy a Model Compiled with Neo (Console) (p. 671)
- Deploy a Model Compiled with Neo (Amazon SageMaker SDK) (p. 676)

Deploy a Model Compiled with Neo (AWS CLI)

The deployment of a Neo-compiled model with the CLI has three steps.

Topics

- Create a Model That Was Compiled with Neo (AWS CLI) (p. 669)
- Create the Endpoint Configuration (AWS CLI) (p. 671)
- Create an Endpoint (AWS CLI) (p. 671)

Create a Model That Was Compiled with Neo (AWS CLI)

For the full syntax of the CreateModel API, see CreateModel (p. 902).

For Neo-compiled models, use one of the following values for PrimaryContainer/ContainerHostname, depending on your region and applications:

- Amazon SageMaker Image Classification
  - 301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest
  - 785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest
  - 007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest
  - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest
- Amazon SageMaker XGBoost
  - 301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest
• 785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest

**TensorFlow**: The TensorFlow version used must be in *TensorFlow SageMaker Estimators* list.

• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3

**MXNet** The MXNet version used must be in *MXNet SageMaker Estimators* list.

• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet: [mxnet-version]-[cpu/gpu]-py3
• 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet: [mxnet-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet: [mxnet-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet: [mxnet-version]-[cpu/gpu]-py3

**Pytorch** The Pytorch version used must be in *Pytorch SageMaker Estimators* list.

• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
• 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3

Also, if you are using **TensorFlow**, **Pytorch**, or **MXNet**, add the following key-value pair to `PrimaryContainer/Environment`:

```
"Environment": {
  "SAGEMAKER_SUBMIT_DIRECTORY" : "[Full S3 path for *.tar.gz file containing the training script]"
}
```

The script must be packaged as a *.tar.gz* file. The *.tar.gz* file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

• `neo_preprocess(payload, content_type)`: Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
• `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the *Amazon SageMaker Neo Sample Notebooks* (p. 664) for examples using these functions.
Create the Endpoint Configuration (AWS CLI)

For the full syntax of the CreateEndpointConfig API, see CreateEndpointConfig (p. 878). You must specify the correct instance type in ProductionVariants/InstanceType. It is imperative that this value matches the instance type specified in your compilation job.

Create an Endpoint (AWS CLI)

For the full syntax of the CreateEndpoint API, see CreateEndpoint (p. 875).

Deploy a Model Compiled with Neo (Console)

You can create a Neo endpoint in the Amazon SageMaker console. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose Models, and then choose Create models from the Inference group. On the Create model page, complete the Model name, IAM role, and, if needed, VPC fields.

To add information about the container used to deploy your model, choose Add container, then choose Next. Complete the Container input options, Location of inference code image, and Location of model artifacts, and optionally, Container host name, and Environmental variables fields.
To deploy Neo-compiled models, choose the following:

- **Container input options**: Provide model artifacts and inference image
- **Location of inference code image**: Choose one of the following images, depending on the region and kind of application:
  - **Amazon SageMaker Image Classification**
    - 301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest
    - 78557368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest
    - 007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest
    - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest
  - **Amazon SageMaker XGBoost**
    - 301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest
    - 78557368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest
    - 007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest
    - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest
  - **TensorFlow**: The TensorFlow version used must be in TensorFlow SageMaker Estimators list.
• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3
• 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow:
  [tensorflow-version]-[cpu/gpu]-py3

**MXNet** The MXNet version used must be in MXNet SageMaker Estimators list.
• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:
  [mxnet-version]-[cpu/gpu]-py3
• 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:
  [mxnet-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:
  [mxnet-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:
  [mxnet-version]-[cpu/gpu]-py3

**Pytorch** The Pytorch version used must be in Pytorch SageMaker Estimators list.
• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:
  [pytorch-version]-[cpu/gpu]-py3
• 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:
  [pytorch-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:
  [pytorch-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:
  [pytorch-version]-[cpu/gpu]-py3

**Location of model artifact**: the full S3 path of the compiled model artifact generated by the Neo compilation API.

**Environmental variables**:
• Omit this field for **SageMaker Image Classification** and **SageMaker XGBoost**.
• For **TensorFlow**, **Pytorch**, and **MXNet**, specify the environment variable
  `SAGEMAKER_SUBMIT_DIRECTORY` as the full S3 path that contains the training script.

The script must be packaged as a *.tar.gz* file. The *.tar.gz* file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

• `neo_preprocess(payload, content_type)`: Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
• `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the Amazon SageMaker Neo Sample Notebooks (p. 664) for examples using these functions.

Confirm that the information for the containers is accurate, and then choose **Create model**. This takes you to the create model landing page. Select the **Create endpoint** button there.
In Create and configure endpoint diagram, specify the Endpoint name. Choose Create a new endpoint configuration in Attach endpoint configuration.

In New endpoint configuration page, specify the Endpoint configuration name.
Then press Edit next to the name of the model and specify the correct Instance type on the Edit Production Variant page. It is imperative that the Instance type value match the one specified in your compilation job.
When you're done click Save, then click Create endpoint configuration on the New endpoint configuration page, and then click Create endpoint.

**Deploy a Model Compiled with Neo (Amazon SageMaker SDK)**

The object handle for the compiled model supplies the deploy function, which allows you to create an endpoint to serve inference requests. The function lets you set the number and type of instances that are used for the endpoint. You must choose an instance for which you have compiled your model. For example, in the job compiled in Compile a Model (Amazon SageMaker SDK) (p. 668) section, this is ml_c5. The Neo API uses a special runtime, the Neo runtime, to run Neo-optimized models.

```python
predictor = compiled_model.deploy(initial_instance_count = 1, instance_type = 'ml.c5.4xlarge')
```

After the command is done, the name of the newly created endpoint is printed in the jupyter notebook.

**Deploy a Model Compiled with Neo (AWS IoT Greengrass)**

AWS IoT Greengrass extends cloud capabilities to local devices. It enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. With AWS IoT Greengrass, you can perform machine learning inference at the edge on locally generated data using cloud-trained models. Currently, you can deploy models on to all AWS IoT Greengrass devices based on ARM® Cortex-A™, Intel® Atom™, and Nvidia® Jetson™ series processors. For more information on deploying a Lambda inference application to perform machine learning inferences with AWS IoT Greengrass, see Perform Machine Learning Inference.
Request Inferences from a Deployed Service

If you have followed instructions in Deploy a Model Compiled with Neo with Hosting Services (p. 669), you should have an Amazon SageMaker endpoint set up and running. You can now submit inference requests using Boto3 client. Here is an example of sending an image for inference:

```python
import boto3
import json

endpoint = '<insert name of your endpoint here>'
runtime = boto3.Session().client('sagemaker-runtime')

# Read image into memory
with open(image, 'rb') as f:
    payload = f.read()
# Send image via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='application/x-image', Body=payload)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

For XGBoost application, you should submit a CSV text instead:

```python
import boto3
import json

endpoint = '<insert your endpoint here>'
runtime = boto3.Session().client('sagemaker-runtime')

csv_text = '1,-1.0,1.0,1.5,2.6'
# Send CSV text via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv', Body=csv_text)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

Note that BYOM allows for a custom content type. For more information, see InvokeEndpoint (p. 1260).

Troubleshooting Neo Compilation Errors

This section contains information about how to understand and prevent common errors, the error messages they generate, and guidance on how to resolve these errors. It also contains lists of the frameworks and the operations in each of those frameworks that Neo supports.

Topics

- Prevent Neo Input Errors (p. 677)
- Neo Error Messages (p. 682)
- Resolve Neo Errors (p. 683)

Prevent Neo Input Errors

Some of the most common errors are due to invalid inputs. This section contains information arranged in question and answer form to help you avoid these errors.

Which frameworks does Neo support?
Which operators does Amazon SageMaker Neo support for these frameworks?

The following table lists the supported operations for each framework.

<table>
<thead>
<tr>
<th>MXNet</th>
<th>TensorFlow</th>
<th>PyTorch/ONNX</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'_add_scalar'</code></td>
<td>'Add'</td>
<td>'Abs'</td>
</tr>
<tr>
<td><code>'_add_symbol'</code></td>
<td>'ArgMax'</td>
<td>'Add'</td>
</tr>
<tr>
<td><code>'_contrib_MultiBoxDetection'</code></td>
<td>'ArgMin'</td>
<td>'ArgMax'</td>
</tr>
<tr>
<td><code>'_contrib_MultiBoxPrior'</code></td>
<td>'AvgPool'</td>
<td>'ArgMin'</td>
</tr>
<tr>
<td><code>'_copy'</code></td>
<td>'BatchNormWithGlobalNormalization'</td>
<td>'AvgPool'</td>
</tr>
<tr>
<td><code>'_div_scalar'</code></td>
<td>'BiasAdd'</td>
<td>'BatchNormalization'</td>
</tr>
<tr>
<td><code>'_div_symbol'</code></td>
<td>'Cast'</td>
<td>'Cast'</td>
</tr>
<tr>
<td><code>'_minus_scalar'</code></td>
<td>'Ceil'</td>
<td>'Ceil'</td>
</tr>
<tr>
<td><code>'_mul_symbol'</code></td>
<td>'CheckNumerics'</td>
<td>'Clip'</td>
</tr>
<tr>
<td><code>'_plus'</code></td>
<td>'Concat'</td>
<td>'Concat'</td>
</tr>
<tr>
<td><code>'_plus_scalar'</code></td>
<td>'Conv2D'</td>
<td>'ConstantFill'</td>
</tr>
<tr>
<td><code>'_pow_scalar'</code></td>
<td>'DecodeJpeg'</td>
<td>'Conv'</td>
</tr>
<tr>
<td><code>'_rdiv_scalar'</code></td>
<td>'DepthwiseConv2dNative'</td>
<td>'ConvTranspose'</td>
</tr>
<tr>
<td><code>'_rminus_scalar'</code></td>
<td>'Elu'</td>
<td>'Div'</td>
</tr>
<tr>
<td><code>'_rpow_scalar'</code></td>
<td>'Equal'</td>
<td>'Dropout'</td>
</tr>
<tr>
<td><code>'_rsub_scalar'</code></td>
<td>'ExpandDims'</td>
<td>'Elu'</td>
</tr>
<tr>
<td><code>'_sub_scalar'</code></td>
<td>'Fill'</td>
<td>'Exp'</td>
</tr>
<tr>
<td><code>'_sub_symbol'</code></td>
<td>'Floor'</td>
<td>'FC'</td>
</tr>
<tr>
<td>'Activation'</td>
<td>'FusedBatchNorm'</td>
<td>'Flatten'</td>
</tr>
<tr>
<td>'add_n'</td>
<td>'FusedBatchNormV2'</td>
<td>'Floor'</td>
</tr>
<tr>
<td>'argmax'</td>
<td>'GatherV2'</td>
<td>'Gather'</td>
</tr>
<tr>
<td>'BatchNorm'</td>
<td>'Greater'</td>
<td>'Gemm'</td>
</tr>
<tr>
<td>'BatchNorm_v1'</td>
<td>'GreaterEqual'</td>
<td>'GlobalAveragePool'</td>
</tr>
<tr>
<td>MXNet</td>
<td>TensorFlow</td>
<td>PyTorch/ONNX</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>'broadcast_add'</td>
<td>'Identity'</td>
<td>'GlobalMaxPool'</td>
</tr>
<tr>
<td>'broadcast_div'</td>
<td>'LeakyRelu'</td>
<td>'HardSigmoid'</td>
</tr>
<tr>
<td>'broadcast_mul'</td>
<td>'Less'</td>
<td>'Identity'</td>
</tr>
<tr>
<td>'broadcast_sub'</td>
<td>'LessEqual'</td>
<td>'ImageScaler'</td>
</tr>
<tr>
<td>'broadcast_to'</td>
<td>'LRN'</td>
<td>'LeakyRelu'</td>
</tr>
<tr>
<td>'cast'</td>
<td>'MatMul'</td>
<td>'Log'</td>
</tr>
<tr>
<td>'Cast'</td>
<td>'Maximum'</td>
<td>'LogSoftmax'</td>
</tr>
<tr>
<td>'clip'</td>
<td>'MaxPool'</td>
<td>'LRN'</td>
</tr>
<tr>
<td>'Concat'</td>
<td>'Mean'</td>
<td>'MatMul'</td>
</tr>
<tr>
<td>'concat'</td>
<td>'Minimum'</td>
<td>'Max'</td>
</tr>
<tr>
<td>'Convolution'</td>
<td>'Mul'</td>
<td>'MaxPool'</td>
</tr>
<tr>
<td>'Convolution_v1'</td>
<td>'NotEqual'</td>
<td>'Mean'</td>
</tr>
<tr>
<td>'Crop'</td>
<td>'Pack'</td>
<td>'Min'</td>
</tr>
<tr>
<td>'Deconvolution'</td>
<td>'Pad'</td>
<td>'Mul'</td>
</tr>
<tr>
<td>'Dropout'</td>
<td>'PadV2'</td>
<td>'Neg'</td>
</tr>
<tr>
<td>'elemwise_add'</td>
<td>'Range'</td>
<td>'Pad'</td>
</tr>
<tr>
<td>'elemwise_div'</td>
<td>'Rank'</td>
<td>'ParametricSoftplus'</td>
</tr>
<tr>
<td>'elemwise_mul'</td>
<td>'Relu'</td>
<td>'Pow'</td>
</tr>
<tr>
<td>'elemwise_sub'</td>
<td>'Relu6'</td>
<td>'PRelu'</td>
</tr>
<tr>
<td>'exp'</td>
<td>'Reshape'</td>
<td>'Reciprocal'</td>
</tr>
<tr>
<td>'expand_dims'</td>
<td>'ResizeBilinear'</td>
<td>'ReduceMax'</td>
</tr>
<tr>
<td>'flatten'</td>
<td>'Rsqrt'</td>
<td>'ReduceMean'</td>
</tr>
<tr>
<td>'Flatten'</td>
<td>'Selu'</td>
<td>'ReduceMin'</td>
</tr>
<tr>
<td>'FullyConnected'</td>
<td>'Shape'</td>
<td>'ReduceProd'</td>
</tr>
<tr>
<td>'LeakyReLU'</td>
<td>'Sigmoid'</td>
<td>'ReduceSum'</td>
</tr>
<tr>
<td>'LinearRegressionOutput'</td>
<td>'Softmax'</td>
<td>'Relu'</td>
</tr>
<tr>
<td>'log'</td>
<td>'Square'</td>
<td>'Reshape'</td>
</tr>
<tr>
<td>'log_softmax'</td>
<td>'Squeeze'</td>
<td>'Scale'</td>
</tr>
<tr>
<td>'LRN'</td>
<td>'StridedSlice'</td>
<td>'ScaledTanH'</td>
</tr>
<tr>
<td>'max'</td>
<td>'Sub'</td>
<td>'Selu'</td>
</tr>
<tr>
<td>'max_axis'</td>
<td>'Sum'</td>
<td>'Shape'</td>
</tr>
<tr>
<td>MXNet</td>
<td>TensorFlow</td>
<td>PyTorch/ONNX</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>'min'</td>
<td>'Tanh'</td>
<td>'Sigmoid'</td>
</tr>
<tr>
<td>'min_axis'</td>
<td>'Transpose'</td>
<td>'Slice'</td>
</tr>
<tr>
<td>'negative'</td>
<td></td>
<td>'Softmax'</td>
</tr>
<tr>
<td>'Pooling'</td>
<td></td>
<td>'SoftPlus'</td>
</tr>
<tr>
<td>'Pooling_v1'</td>
<td></td>
<td>'Softsign'</td>
</tr>
<tr>
<td>'relu'</td>
<td></td>
<td>'SpatialBN'</td>
</tr>
<tr>
<td>'Reshape'</td>
<td></td>
<td>'Split'</td>
</tr>
<tr>
<td>'reshape'</td>
<td></td>
<td>'Sqrt'</td>
</tr>
<tr>
<td>'reshape_like'</td>
<td></td>
<td>'Squeeze'</td>
</tr>
<tr>
<td>'sigmoid'</td>
<td></td>
<td>'Sub'</td>
</tr>
<tr>
<td>'slice_like'</td>
<td></td>
<td>'Sum'</td>
</tr>
<tr>
<td>'SliceChannel'</td>
<td></td>
<td>'Tanh'</td>
</tr>
<tr>
<td>'softmax'</td>
<td></td>
<td>'ThresholdedRelu'</td>
</tr>
<tr>
<td>'Softmax'</td>
<td></td>
<td>'Transpose'</td>
</tr>
<tr>
<td>'SoftmaxActivation'</td>
<td></td>
<td>'Unsqueeze'</td>
</tr>
<tr>
<td>'SoftmaxOutput'</td>
<td></td>
<td>'Upsample'</td>
</tr>
<tr>
<td>'split'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'sum'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'sum_axis'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'tanh'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'transpose'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'UpSampling'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Which model architectures does Neo support?**

Neo supports image classification models.

**Which model format files does Neo read in?**

The file needs to be formatted as a tar.gz file that includes additional files that depend on the type of framework.

- **TensorFlow**: Neo supports saved models and frozen models.
  
  For **saved models**, Neo expects one .pb or one .pbtxt file and a variables directory that contains variables.

  For **frozen models**, Neo expect only one .pb or .pbtxt file.

- **Keras**: Neo expects one .h5 file containing the model definition.
• **PyTorch**: Neo expects one .pth file containing the model definition.
• **MXNET**: Neo expects one symbol file (.json) and one parameter file (.params).
• **XGBoost**: Neo expects one XGBoost model file (.model) where the number of nodes in a tree can't exceed \(2^{31}\).
• **ONNX**: Neo expects one .onnx file.

**What input data shapes does Neo expect?**

Neo expects the name and shape of the expected data inputs for your trained model with a JSON dictionary form or list form. The data inputs are framework specific.

• **TensorFlow**: You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"input":[1,1024,1024,3]}`
    - If using the CLI, `{\"input\":[1,1024,1024,3]}`
  - Examples for two inputs:
    - If using the console, `{"data1": [1,28,28,1], "data2": [1,28,28,1]}`
    - If using the CLI, `{\"data1\": [1,28,28,1], \"data2\": [1,28,28,1]}`

• **Keras**: You must specify the name and shape (NCHW format) of expected data inputs using a dictionary format for your trained model. Note that while Keras model artifacts should be uploaded in NHWC (channel-last) format, `DataInputConfig` should be specified in NCHW (channel-first) format. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"input_1": [1,3,224,224]}`
    - If using the CLI, `{\"input_1\": [1,3,224,224]}`
  - Examples for two inputs:
    - If using the console, `{"input_1": [1,3,224,224], "input_2": [1,3,224,224]}`
    - If using the CLI, `{\"input_1\": [1,3,224,224], \"input_2\": [1,3,224,224]}`

• **MXNET/ONNX**: You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, `{"data": [1,3,1024,1024]}`
    - If using the CLI, `{\"data\": [1,3,1024,1024]}`
  - Examples for two inputs:
    - If using the console, `{"var1": [1,1,28,28], "var2": [1,1,28,28]}`
    - If using the CLI, `{\"var1\": [1,1,28,28], \"var2\": [1,1,28,28]}`

• **PyTorch**: You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
  - Examples for one input in dictionary format:
    - If using the console, `{"input0": [1,3,224,224]}`
    - If using the CLI, `{\"input0\": [1,3,224,224]}`
  - Example for one input in list format: `[1,3,224,224]`
  - Examples for two inputs in dictionary format:
    - If using the console, `{"input0": [1,3,224,224], "input1": [1,3,224,224]}`
• If using the CLI,{"input0":[1,3,224,224], "input1":[1,3,224,224]}
• Example for two inputs in list format: [[1,3,224,224], [1,3,224,224]]
• XGBOOST: input data name and shape are not needed.

Neo Error Messages

This section lists and classifies Neo errors and error messages.

Neo Error Messages

This list catalogs the user and system error messages you can receive from Neo deployments.

• User error messages
  • Client permission error: Neo passes the errors for these straight through from the dependent service.
    
    Access Denied when calling sts:AssumeRole
  
  Any 400 error when calling S3 to download or upload a client model.

  PassRole error
  • Load error: Keywords in error messages, 'InputConfiguration','ModelSizeTooBig'.
    
    Load Error: InputConfiguration: Exactly one {.xxx} file is allowed for {yyy} model.
  
  Load Error: ModelSizeTooBig: number of nodes in a tree can't exceed 2^31
  • Compilation error: Keywords in error messages, 'OperatorNotImplemented','OperatorAttributeNotImplemented','OperatorAttributeRequired','OperatorAttributeValueNotValid'.
    
    OperatorNotImplemented: {xxx} is not supported.
  
    OperatorAttributeNotImplemented: {xxx} is not supported in {yyy}.
  
    OperatorAttributeRequired: Required attribute {xxx} not found in {yyy}.
  
    OperatorAttributeValueNotValid: The value of attribute {xxx} in operator {yyy} cannot be negative.
  • Any Malformed Input Errors
  • System error messages
    For system errors, Neo shows only one error message similar to the following: There was an unexpected error during compilation, check your inputs and try again in a few minutes.
    • This covers all unexpected errors and errors that are not user errors.

Neo Error Classifications

This list classifies the user errors you can receive from Neo. These include access and permission errors and load errors for each of the supported frameworks. All other errors are system errors.

• Client permission error: Neo passes the errors for these straight through from the dependent service.
    
    Access Denied when calling sts:AssumeRole
  
  Any 400 error when calling Amazon S3 to download or upload a client model.
• **Load error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, check whether the tarball contains the necessary files for compilation. The checking criteria is framework-specific:
  - **TensorFlow**: Expects only protobuf file (*.pb or *.pbtxt). For saved models, expects one variables folder.
  - **Pytorch**: Expect only one pytorch file (*.pth).
  - **MXNET**: Expect only one symbol file (*.json) and one parameter file (*.params).
  - **XGBoost**: Expect only one XGBoost model file (*.model). The input model has size limitation.
• **Compilation error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, and that the tarball contains necessary files for compilation. The checking criteria is:
  - **OperatorNotImplemented**: An operator has not been implemented.
  - **OperatorAttributeNotImplemented**: The attribute in the specified operator has not been implemented.
  - **OperatorAttributeRequired**: An attribute is required for an internal symbol graph, but it is not listed in the user input model graph.
  - **OperatorAttributeValueNotValid**: The value of the attribute in the specific operator is not valid.

### Resolve Neo Errors

This section provides guidance on troubleshooting common issues with Neo. These include permission, load, compilation, and system errors and errors involving invalid inputs and unsupported operations.

- **Catalog of Known Issues**:
  - If you see **Client Permission Error**, review the set up documentation and make sure that you have correctly granted the permissions that are failing.
  - If you see **Load Error**, check the model format files that Neo expects for different frameworks.
  - If you see **Compilation Error**, check and address the details error message in your input model graph.
  - If you see **System Error**, try again in a few minutes. If that fails, file a ticket.
  - **Lack of Roles and Permissions**: Review the set up documentation and make sure that you have correctly granted the permissions that are failing.
  - **Invalid API and Console Inputs**: Fix your input as described in the validation error.
  - **Unsupported Operators**:
    - Check the failure reason where Neo has listed all unsupported operators with the keyword ‘OperatorNotImplemented’.
    - For example: Compilation Error: OperatorNotImplemented: The following operators are not implemented: {'_sample_multinomial', 'RNN'}
    - Remove the unsupported operators from your input model graph and test it again.

### Use Amazon SageMaker Elastic Inference (EI)

By using Amazon Elastic Inference (EI), you can speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models, but at a fraction of the cost of using a GPU instance for your endpoint. EI allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance. Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type, and then add that model as a production variant to an endpoint configuration that you use to deploy a hosted endpoint. You can also add an EI accelerator to a Amazon SageMaker notebook instance so that you can test and evaluate inference performance when you are building your models.
Elastic Inference is supported in EI-enabled versions of TensorFlow and MXNet. To use any other deep learning framework, export your model by using ONNX, and then import your model into MXNet. You can then use your model with EI as an MXNet model. For information about importing an ONNX model into MXNet, see https://mxnet.apache.org/api/python/docs/tutorials/packages/onnx/super_resolution.html.

Topics
- How EI Works (p. 684)
- Choose an EI Accelerator Type (p. 684)
- Use EI in a Amazon SageMaker Notebook Instance (p. 685)
- Use EI on a Hosted Endpoint (p. 685)
- Frameworks that Support EI (p. 685)
- Use EI with Amazon SageMaker Built-in Algorithms (p. 685)
- EI Sample Notebooks (p. 686)
- Set Up to Use EI (p. 686)
- Attach EI to a Notebook Instance (p. 689)
- Use EI on Amazon SageMaker Hosted Endpoints (p. 689)

How EI Works

EI accelerators are network attached devices that work along with EC2 instances in your endpoint to accelerate your inference calls. When your model is deployed as an endpoint, ML frameworks use a combination of EC2 instance and accelerator resources to execute inference calls.

The following EI accelerator types are available. You can configure your endpoints or notebook instances with any EI accelerator type.

In the table, the throughput in teraflops (TFLOPS) is listed for both single-precision floating-point (F32) and half-precision floating-point (F16) operations. The memory in GB is also listed.

<table>
<thead>
<tr>
<th>Accelerator Type</th>
<th>F32 Throughput in TFLOPS</th>
<th>F16 Throughput in TFLOPS</th>
<th>Memory in GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.eia2.medium</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>2</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>4</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>2</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>4</td>
<td>32</td>
<td>4</td>
</tr>
</tbody>
</table>

Choose an EI Accelerator Type

Consider the following factors when choosing an accelerator type for a hosted model:

- Models, input tensors and batch sizes influence the amount of accelerator memory you need. Start with an accelerator type that provides at least as much memory as the file size of your trained model.
- Demands on CPU compute resources, GPU-based acceleration, and CPU memory vary significantly between different kinds of deep learning models. The latency and throughput requirements of the
application also determine the amount of compute and acceleration you need. Thoroughly test
different configurations of instance types and EI accelerator sizes to make sure you choose the
configuration that best fits the performance needs of your application.

Use EI in a Amazon SageMaker Notebook Instance

Typically, you build and test machine learning models in a Amazon SageMaker notebook before you
deploy them for production. You can attach EI to your notebook instance when you create the notebook
instance. You can set up an endpoint that is hosted locally on the notebook instance by using the local
mode supported by TensorFlow and MXNet estimators and models in the Amazon SageMaker Python
SDK to test inference performance. For instructions on how to attach EI to a notebook instance and set
up a local endpoint for inference, see Attach EI to a Notebook Instance (p. 689).

Use EI on a Hosted Endpoint

When you are ready to deploy your model for production to provide inferences, you create a Amazon
SageMaker hosted endpoint. You can attach EI to the instance where your endpoint is hosted to increase
its performance at providing inferences. For instructions on how to attach EI to a hosted endpoint
instance, see Use EI on Amazon SageMaker Hosted Endpoints (p. 691).

Frameworks that Support EI

EI is designed to be used with AWS enhanced versions of TensorFlow or Apache MXNet machine learning
frameworks. These enhanced versions of the frameworks are automatically built into containers when
you use the Amazon SageMaker Python SDK, or you can download them as binary files and import
them in your own Docker containers. You can download the EI-enabled binary for TensorFlow from
For information about building a container that uses the EI-enabled version of TensorFlow, see
https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-
tensorflow-serving-container. You can download the EI-enabled binary for Apache MXNet from the
For information about building a container that uses the EI-enabled version of MXNet, see https://
github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-
container.

To use EI in a hosted endpoint, you can use any of the following, depending on your needs.

- SageMaker Python SDK TensorFlow - if you want to use TensorFlow and you don't need to build a
custom container.
- SageMaker Python SDK MXNet - if you want to use MXNet and you don't need to build a custom
container.
- The low-level AWS Amazon SageMaker SDK for Python (Boto 3) - if you need to build a custom
container.

Typically, you don't need to create a custom container unless your model is very complex and requires
extensions to a framework that the Amazon SageMaker pre-built containers do not support.

Use EI with Amazon SageMaker Built-in Algorithms

Currently, the Image Classification Algorithm (p. 271) and Object Detection Algorithm (p. 365) built-in
algorithms support EI. For an example that uses the Image Classification algorithm with EI, see https://
github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/
imageclassification_caltech/image-classification-fulltraining.ipynb.
EI Sample Notebooks

The following Sample notebooks provide examples of using EI in Amazon SageMaker:


Set Up to Use EI

Use the instructions in this topic only if one of the following applies to you:

- You want to use a customized role or permission policy.
- You want to use a VPC for your hosted model or notebook instance.

**Note**
If you already have an execution role that has the AmazonSageMakerFullAccess managed policy attached (this is true for any IAM role that you create when you create a notebook instance, training job, or model in the console) and you are not connecting to an EI model or notebook instance in a VPC, you do not need to make any of these changes to use EI in Amazon SageMaker.

**Topics**
- Set Up Required Permissions (p. 686)
- Use a Custom VPC to Connect to EI (p. 688)

Set Up Required Permissions

To use EI in Amazon SageMaker, the role that you use to open a notebook instance or create a deployable model must have a policy with the required permissions attached. You can attach the AmazonSageMakerFullAccess managed policy, which contains the required permissions, to the role, or you can add a custom policy that has the required permissions. For information about creating an IAM role, see Creating a Role for an AWS Service (Console) in the AWS Identity and Access Management User Guide. For information about attaching a policy to a role, see Adding and Removing IAM Policies.

Add these permissions specifically for connecting EI in an IAM policy:

```json
{
  "Effect": "Allow",
  "Action": [
    "elastic-inference:Connect",
    "ec2:DescribeVpcEndpoints"
  ]
}
```
The following IAM policy is the complete list of required permissions to use EI in Amazon SageMaker:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elastic-inference:Connect",
        "ec2:DescribeVpcEndpoints"
      ],
      "Resource": "*
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:*"
      ],
      "Resource": "*
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "cloudwatch:PutMetricData",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DeleteAlarms",
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:DeleteScheduledAction",
        "application-autoscaling:DeregisterScalableTarget",
        "application-autoscaling:DescribeScalableTargets",
        "application-autoscaling:DescribeScalingActivities",
        "application-autoscaling:DescribeScalingPolicies",
        "application-autoscaling:DescribeScheduledActions",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:PutScheduledAction",
        "application-autoscaling:RegisterScalableTarget",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "logs:PutLogEvents"
      ],
      "Resource": "*
    },
  ]
}
```
Use a Custom VPC to Connect to EI

To use EI with Amazon SageMaker in a VPC, you need to create and configure two security groups, and set up a PrivateLink VPC interface endpoint. EI uses VPC interface endpoint to communicate with
Amazon SageMaker endpoints in your VPC. The security groups you create are used to connect to the VPC interface endpoint.

**Set up Security Groups to Connect to EI**

To use EI within a VPC, you need to create two security groups:

- A security group to control access to the VPC interface endpoint that you will set up for EI.
- A security group that allows Amazon SageMaker to call into the first security group.

Complete the following steps to configure the two security groups:

1. Create a security group with no outbound connections. You will attach this to the VPC endpoint interface you create in the next section.
2. Create a second security group with no inbound connections, but with an outbound connection to the first security group.
3. Edit the first security group to allow inbound connections only to the second security group an all outbound connections.

For more information about VPC security groups, see Security Groups for Your VPC in the Amazon Virtual Private Cloud User Guide.

**Set up a VPC Interface Endpoint to Connect to EI**

To use EI with Amazon SageMaker in a custom VPC, you need to set up a VPC interface endpoint (PrivateLink) for the EI service.

- Set up a VPC interface endpoint (PrivateLink) for the EI. Follow the instructions at Creating an Interface Endpoint. In the list of services, choose `com.amazonaws.<region>.elastic-inference.runtime`. For Security group, make sure you select the first security group you created in the previous section to the endpoint.
- When you set up the interface endpoint, choose all of the Availability Zones where EI is available. EI fails if you do not set up at least two Availability Zones. For information about VPC subnets, see VPCs and Subnets.

**Attach EI to a Notebook Instance**

To test and evaluate inference performance using EI, you can attach EI to a notebook instance when you create or update a notebook instance. You can then use EI in local mode to host a model at an endpoint hosted on the notebook instance. You should test various sizes of notebook instances and EI accelerators to evaluate the configuration that works best for your use case.

**Set Up to Use EI**

To use EI locally in a notebook instance, create a notebook instance with an EI instance. To do this:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker
2. In the navigation pane, choose Notebook instances.
3. Choose Create notebook instance.
4. For Notebook instance name, provide a unique name for your notebook instance.
5. For notebook instance type, provide a CPU instance such as ml.t2.medium.
6. For Elastic Inference (EI), choose an instance from the list, such as ml.eia2.medium.
7. For **IAM role**, choose an IAM role that has the required permissions to use Amazon SageMaker and EI.

8. (Optional) For **VPC - Optional**, if you want the notebook instance to use a VPC, choose one from the available list, otherwise leave it as **No VPC**. If you use a VPC follow the instructions at Use a Custom VPC to Connect to EI (p. 688).

9. (Optional) For **Lifecycle configuration - optional**, either leave it as **No configuration** or choose a lifecycle configuration. For more information, see Customize a Notebook Instance (p. 206).

10. (Optional) For **Encryption key - optional**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.

11. (Optional) For **Volume Size In GB - optional**, leave the default value of 5.

12. (Optional) For **Tags**, add tags to the notebook instance. A tag is a label you assign to help manage your notebook instances. A tag consists of a key and a value both of which you define.

13. Choose **Create Notebook Instance**.

After you create your notebook instance with EI attached, you can create a Jupyter notebook and set up an EI endpoint that is hosted locally on the notebook instance.

**Topics**
- Use EI in Local Mode in Amazon SageMaker (p. 690)

**Use EI in Local Mode in Amazon SageMaker**

To use EI locally in an endpoint hosted on a notebook instance, use local mode with the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet estimators or models. For more information about local mode support in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

**Topics**
- Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models (p. 690)
- Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models (p. 691)

**Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models**

To use EI with TensorFlow in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about Amazon SageMaker Python SDK TensorFlow estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

The following code shows how to use local mode with an estimator object. To call the `deploy` method, you must have previously either:

- Trained the model by calling the `fit` method of an estimator.
- Pass a model artifact when you initialize the model object.

```python
# Deploys the model to a local endpoint
tf_predictor = tf_model.deploy(initial_instance_count=1,
                                instance_type='local',
                                accelerator_type='local_sagemaker_notebook')
```
For a complete example of using EI in local mode with TensorFlow, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb

Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models

To use EI with MXNet in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator object. For more information about Amazon SageMaker Python SDK MXNet estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

The following code shows how to use local mode with an estimator object. You must have previously called the `fit` method of the estimator to train the model.

```python
# Deploys the model to a local endpoint
mxnet_predictor = mxnet_estimator.deploy(initial_instance_count=1,
instance_type='local',
accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with MXNet, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb.

Use EI on Amazon SageMaker Hosted Endpoints

To use Elastic Inference (EI) in Amazon SageMaker with a hosted endpoint for real-time inference, specify an EI accelerator when you create the deployable model to be hosted at that endpoint. You can do this in one of the following ways:

- Use the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet and the Amazon SageMaker pre-built containers for TensorFlow and MXNet
- Build your own container, and use the low-level Amazon SageMaker API (Boto 3). You will need to import the EI-enabled version of either TensorFlow or MXNet from the provided Amazon S3 locations into your container, and use one of those versions to write your training script.
- Use either the Image Classification Algorithm (p. 271) or Object Detection Algorithm (p. 365) built-in algorithms, and use Boto 3 to run your training job and create your deployable model and hosted endpoint.

Topics

- Use EI with an Amazon SageMaker TensorFlow Container (p. 691)
- Use EI with an Amazon SageMaker MXNet Container (p. 692)
- Use EI with Your Own Container (p. 693)

Use EI with an Amazon SageMaker TensorFlow Container

To use TensorFlow with EI in Amazon SageMaker, you need to call the `deploy` method of either the `Estimator` or `Model` objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using TensorFlow in the Amazon SageMaker Python SDK, see: https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.
Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

**Use an Estimator Object**

To use an estimator object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                          instance_type='ml.m4.xlarge',
                          accelerator_type='ml.eia2.medium')
```

**Use a Model Object**

To use a model object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                          instance_type='ml.m4.xlarge',
                          accelerator_type='ml.eia2.medium')
```

**Use EI with an Amazon SageMaker MXNet Container**

To use MXNet with EI in Amazon SageMaker, you need to call the `deploy` method of either the `Estimator` or `Model` objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using MXNet in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

**Use an Estimator Object**

To use an estimator object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                          instance_type='ml.m4.xlarge',
                          accelerator_type='ml.eia2.medium')
```

**Use a Model Object**

To use a model object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                          instance_type='ml.m4.xlarge',
                          accelerator_type='ml.eia2.medium')
```
For a complete example of using EI with MXNet in Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb

Use EI with Your Own Container

To use EI with a model in a custom container that you build, use the low-level Amazon SageMaker SDK for Python (Boto 3). Download and import the AWS EI-enabled versions of TensorFlow or Apache MXNet machine learning frameworks, and write your training script using those frameworks.

Import the EI Version of TensorFlow or MXNet into Your Docker Container

To use EI with your own container, you need to import either the Amazon EI TensorFlow Serving library or the Amazon EI Apache MXNet library into your container. The EI-enabled versions of TensorFlow and MXNet are currently available as binary files stored in Amazon S3 locations. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-tensorflow. For information about building a container that uses the EI-enabled version of TensorFlow, see https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at https://s3.console.aws.amazon.com/s3/buckets/amazonei-apachemxnet. For information about building a container that uses the EI-enabled version of MXNet, see https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container.

Create an EI Endpoint with Boto 3

To create an endpoint by using Boto 3, you first create an endpoint configuration. The endpoint configuration specifies one or more models (called production variants) that you want to host at the endpoint. To attach EI to one or more of the production variants hosted at the endpoint, you specify one of the EI instance types as the AcceleratorType field for that ProductionVariant. You then pass that endpoint configuration when you create the endpoint.

Create an Endpoint Configuration

To use EI, you need to specify an accelerator type in the endpoint configuration:

```python
# Create Endpoint Configuration from time import gmtime, strftime
evaluation_config_name = 'ImageClassificationEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(evaluation_config_name)
create_evaluation_config_response = sagemaker.create_endpoint_config(
    EndpointConfigName = evaluation_config_name,
    ProductionVariants=[
        'InstanceType':'ml.m4.xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic',
        'AcceleratorType':'ml.eia2.medium'])
print("Evaluation Config Arn: ") + create_evaluation_config_response['EndpointConfigArn']
```

Create an Endpoint

After you create an endpoint configuration with an accelerator type, you can proceed to create an endpoint.
endpoint_name = 'ImageClassificationEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
endpoint_response = sagemaker.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)

After the endpoint is created you can invoke it using the invoke_endpoint method in a boto3 runtime object as you would any other endpoint.

Automatically Scale Amazon SageMaker Models

Amazon SageMaker supports automatic scaling for production variants. **Automatic scaling** dynamically adjusts the number of instances provisioned for a production variant in response to changes in your workload. When the workload increases, automatic scaling brings more instances online. When the workload decreases, automatic scaling removes unnecessary instances so that you don’t pay for provisioned variant instances that you aren’t using.

To use automatic scaling for a production variant, you define and apply a scaling policy that uses Amazon CloudWatch metrics and target values that you assign. Automatic scaling uses the policy to adjust the number of instances up or down in response to actual workloads.

You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A *predefined metric* is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Alternatively, you can use either the AWS Command Line Interface (AWS CLI) or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric. We strongly recommend that you load test your automatic scaling configuration to ensure that it works correctly before using it to manage production traffic.

For information about deploying trained models as endpoints, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35).

**Topics**
- Automatic Scaling Components (p. 694)
- Before You Begin (p. 697)
- Related Topics (p. 697)
- Configure Automatic Scaling for a Production Variant (p. 697)
- Edit a Scaling Policy (p. 703)
- Delete a Scaling Policy (p. 704)
- Update Endpoints that Use Automatic Scaling (p. 705)
- Load Testing for Production Variant Automatic Scaling (p. 706)
- Best Practices for Configuring Automatic Scaling (p. 707)

**Automatic Scaling Components**

To adjust the number of instances hosting a production variant, Amazon SageMaker automatic scaling uses a scaling policy. Automatic scaling has the following components:

- **Required permissions**—Permissions that are required to perform automatic scaling actions.
- **A service-linked role**—An AWS Identity and Access Management (IAM) role that is linked to a specific AWS service. A service-linked role includes all of the permissions that the service requires to call other AWS services on your behalf. Amazon SageMaker automatic scaling automatically generates this role, AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint, for you.
Automatic Scaling Components

- A target metric—The Amazon CloudWatch metric that Amazon SageMaker automatic scaling uses to determine when and how much to scale.
- Minimum and maximum capacity—The minimum and maximum number of instances to use for scaling the variant.
- A cool down period—The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.

Required Permissions for Automatic Scaling

The SagemakerFullAccessPolicy IAM policy has all of the permissions required to perform automatic scaling actions. For more information about Amazon SageMaker IAM roles, see Amazon SageMaker Roles (p. 758).

If you are using a custom permission policy, you must include the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:UpdateEndpointWeightsAndCapacities"
    ],
    "Resource": "*

    
    "Action": [
        "application-autoscaling:*
    ],
    "Effect": "Allow",
    "Resource": "*"

    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource":
    "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": { "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com" }
    }
}

{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DeleteAlarms"
    ],
    "Resource": "*"
}
```

Service-Linked Role for Automatic Scaling

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all of the permissions that
the service requires to call other AWS services on your behalf. Automatic scaling uses the
AWS
dServiceRoleForApplicationAutoScaling_SageMakerEndpoint service-linked role. For more
information, see Service-Linked Roles for Application Auto Scaling in the Application Auto Scaling User
Guide.

**Target Metric for Automatic Scaling**

Amazon SageMaker automatic scaling uses target-tracking scaling policies. You configure the target-
tracking scaling policy by specifying a predefined or custom metric and a target value for the metric. For
more information, see Target Tracking Scaling Policies.

Amazon CloudWatch alarms trigger the scaling policy, which calculate how to adjust scaling based
on the metric and target value that you set. The scaling policy adds or removes endpoint instances
as required to keep the metric at, or close to, the specified target value. In addition, a target-tracking
scaling policy also adjusts to fluctuations in the metric when a workload changes. The policy minimizes
rapid fluctuations in the number of available instances for your variant.

For example, a scaling policy that uses the predefined InvocationsPerInstance metric with a target
value of 70 can keep InvocationsPerInstance at, or close to 70.

**Minimum and Maximum Capacity for Automatic Scaling**

You can specify the maximum number of endpoint instances that Application Auto Scaling manages for
the variant. The maximum value must be equal to or greater than the value specified for the minimum
number of endpoint instances. Amazon SageMaker automatic scaling does not enforce a limit for this
value.

You can also specify the minimum number of instances that Application Auto Scaling manages for the
variant. This value must be at least 1, and equal to or less than the value specified for the maximum
number of variant instances.

To determine the minimum and maximum number of instances that you need for typical traffic, test your
automatic scaling configuration with the expected rate of traffic to your variant.

**Cooldown Period for Automatic Scaling**

Tune the responsiveness of a target-tracking scaling policy by adding a cooldown period. A cooldown
period controls when your variant is scaled in and out by blocking subsequent scale-in or scale-out
requests until the period expires. This slows the deletion of variant instances for scale-in requests, and
the creation of variant instances for scale-out requests. A cooldown period helps to ensure that it doesn't
launch or terminate additional instances before the previous scaling activity takes effect. After automatic
scaling dynamically scales using a scaling policy, it waits for the cooldown period to complete before
resuming scaling activities.

You configure the cooldown period in your automatic scaling policy. You can specify the following
cooldown periods:

- A scale-in activity reduces the number of variant instances. A scale-in cooldown period specifies the
  amount of time, in seconds, after a scale-in activity completes before another scale-in activity can
  start.
- A scale-out activity increases the number of variant instances. A scale-out cooldown period specifies
  the amount of time, in seconds, after a scale-out activity completes before another scale-out activity
  can start.

If you don't specify a scale-in or a scale-out cooldown period automatic scaling use the default, which is
300 seconds for each.
If instances are being added or removed too quickly when you test your automatic scaling configuration, consider increasing this value. You can see this behavior if the traffic to your variant has a lot of spikes, or if you have multiple automatic scaling policies defined for a variant.

If instances are not being added quickly enough to address increased traffic, consider decreasing this value.

**Before You Begin**

Before you can use automatically scaled model deployment, create an Amazon SageMaker model deployment. For more information about deploying a model endpoint, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35).

When automatic scaling adds a new variant instance, it is the same instance class as the one used by the primary instance.

**Related Topics**

- What Is Application Auto Scaling?

**Configure Automatic Scaling for a Production Variant**

You can configure automatic scaling for a variant with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

**Topics**

- Configure Automatic Scaling for a Production Variant (Console) (p. 697)
- Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API) (p. 698)

**Configure Automatic Scaling for a Production Variant (Console)**

**To configure automatic scaling for a production variant (console)**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint that you want to configure.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. For **Endpoint runtime settings**, choose **Configure auto scaling**.

   The **Configure variant automatic scaling** page appears.
6. For **Minimum capacity**, type the minimum number of instances that you want the scaling policy to maintain. At least 1 instance is required.
7. For **Maximum capacity**, type the maximum number of instances that you want the scaling policy to maintain.
8. For the target value, type the average number of invocations per instance per minute for the variant. To determine this value, follow the guidelines in Load Testing (p. 706).

   Application Auto Scaling adds or removes instances to keep the metric close to the value that you specify.
9. For **Scale-in cool down (seconds)** and **Scale-out cool down (seconds)**, type the number seconds for each cool down period. Assuming that the order in the list is based on either most important to less important of first applied to last applied.

10. Select **Disable scale in** to prevent the scaling policy from deleting variant instances if you want to ensure that your variant scales out to address increased traffic, but are not concerned with removing instances to reduce costs when traffic decreases, disable scale-in activities.

   Scale-out activities are always enabled so that the scaling policy can create endpoint instances as needed.

11. Choose **Save**.

This procedure registers a variant as a scalable target with Application Auto Scaling. When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a burstable performance instance such as T2

**Note**  
Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because they already allow for increased capacity under increased workloads. For information about burstable performance instances, see [Amazon EC2 Instance Types](https://docs.aws.amazon.com/AmazonSageMaker/latest/DG/custom-scaling.html).

### Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API)

With the AWS CLI or the Application Auto Scaling API, you can configure automatic scaling based on either a predefined or a custom metric.

**Register a Production Variant**

To define the scaling limits for the variant, register your variant with Application Auto Scaling. Application Auto Scaling dynamically scales the number of variant instances.

To register your variant, you can use either the AWS CLI or the Application Auto Scaling API.

When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant resource exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a Burstable Performance Instance such as T2

**Note**  
Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because burstable instances already allow for increased capacity under increased workloads. For information about Burstable Performance Instances, see [Amazon EC2 Instance Types](https://docs.aws.amazon.com/AmazonSageMaker/latest/DG/custom-scaling.html).

**Register a Production Variant (AWS CLI)**

To register your endpoint, use the `register-scalable-target` AWS CLI command with the following parameters:
• --service-namespace—Set this value to sagemaker.
• --resource-id—The resource identifier for the production variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example endpoint/MyEndPoint/variant/MyVariant.
• --scalable-dimension—Set this value to sagemaker:variant:DesiredInstanceCount.
• --min-capacity—The minimum number of instances that Application Auto Scaling must manage for this endpoint. Set min-capacity to at least 1. It must be equal to or less than the value specified for max-capacity.
• --max-capacity—The maximum number of instances that Application Auto Scaling should manage. Set max-capacity to a minimum of 1, it must be equal to or greater than the value specified for min-capacity.

Example

The following example shows how to register an endpoint variant named MyVariant that is dynamically scaled to have one to eight instances:

```bash
aws application-autoscaling register-scalable-target 
  --service-namespace sagemaker 
  --resource-id endpoint/MyEndPoint/variant/MyVariant 
  --scalable-dimension sagemaker:variant:DesiredInstanceCount 
  --min-capacity 1 
  --max-capacity 8
```

Register a Production Variant (Application Auto Scaling API)

To register your endpoint variant with Application Auto Scaling, use the RegisterScalableTarget Application Auto Scaling API action with the following parameters:

• ServiceNamespace—Set this value to sagemaker.
• ResourceID—The resource identifier for the production variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant, for example endpoint/MyEndPoint/variant/MyVariant.
• ScalableDimension—Set this value to sagemaker:variant:DesiredInstanceCount.
• MinCapacity—The minimum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or less than the value specified for MaxCapacity.
• MaxCapacity—The maximum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or greater than the value specified for MinCapacity.

Example

The following example shows how to register an Amazon SageMaker production variant that is dynamically scaled to use one to eight instances:

```plaintext
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
```
Define a Target-Tracking Scaling Policy

To specify the metrics and target values for a scaling policy, you configure a target-tracking scaling policy. You can use either a predefined metric or a custom metric.

Scaling policy configuration is represented by a JSON block. You save your scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

The following options are available for defining a target-tracking scaling policy configuration.

Topics
- Use a Predefined Metric (p. 700)
- Use a Custom Metric (p. 700)
- Add a Cooldown Period (p. 701)
- Disable Scale-in Activity (p. 701)

Use a Predefined Metric

To quickly define a target-tracking scaling policy for a variant, use the SageMakerVariantInvocationsPerInstance predefined metric. SageMakerVariantInvocationsPerInstance is the average number of times per minute that each instance for a variant is invoked. We strongly recommend using this metric.

To use a predefined metric in a scaling policy, create a target tracking configuration for your policy. In the target tracking configuration, include a PredefinedMetricSpecification for the predefined metric and a TargetValue for the target value of that metric.

Example

The following example is a typical policy configuration for target-tracking scaling for a variant. In this configuration, we use the SageMakerVariantInvocationsPerInstance predefined metric to adjust the number of variant instances so that each instance has an InvocationsPerInstance metric of 70.

```
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    }
}
```

Use a Custom Metric

If you need to define a target-tracking scaling policy that meets your custom requirements, define a custom metric. You can define a custom metric based on any production variant metric that changes in proportion to scaling.

Not all Amazon SageMaker metrics work for target tracking. The metric must be a valid utilization metric, and it must describe how busy an instance is. The value of the metric must increase or decrease
in inverse proportion to the number of variant instances. That is, the value of the metric should decrease when the number of instances increases.

**Important**
Before deploying automatic scaling in production, you must test automatic scaling with your custom metric.

**Example**
The following example is a target-tracking configuration for a scaling policy. In this configuration, for a variant named `my-variant`, a custom metric adjusts the variant based on an average CPU utilization of 50 percent across all instances.

```json
{
  "TargetValue": 50,
  "CustomizedMetricSpecification": {
    "MetricName": "CPUUtilization",
    "Namespace": "/aws/sagemaker/Endpoints",
    "Dimensions": [
      {"Name": "EndpointName", "Value": "my-endpoint" },
      {"Name": "VariantName", "Value": "my-variant" }
    ],
    "Statistic": "Average",
    "Unit": "Percent"
  }
}
```

**Add a Cooldown Period**
To add a cooldown period for scaling out your variant, specify a value, in seconds, for `ScaleOutCooldown`. Similarly, to add a cooldown period for scaling in your variant, add a value, in seconds, for `ScaleInCooldown`. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see [TargetTrackingScalingPolicyConfiguration](#) in the [Application Auto Scaling API Reference](#).

**Example**
The following is an example of a target-tracking policy configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric is used to adjust a variant based on an average of 70 across all instances of that variant. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```json
{
  "TargetValue": 70.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
  },
  "ScaleInCooldown": 600,
  "ScaleOutCooldown": 300
}
```

**Disable Scale-in Activity**
You can prevent the target-tracking scaling policy configuration from scaling in your variant by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting instances, while still allowing it to create them as needed.

To enable or disable scale-in activity for your variant, specify a Boolean value for `DisableScaleIn`. For more information about `DisableScaleIn`, see [TargetTrackingScalingPolicyConfiguration](#) in the [Application Auto Scaling API Reference](#).
Example

The following is an example of a target-tracking configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric adjusts a variant based on an average of 70 across all instances of that variant. The configuration disables scale-in activity for the scaling policy.

```json
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",
        "DisableScaleIn": true
    }
}
```

Apply a Scaling Policy to a Production Variant

After registering your variant and defining a scaling policy, apply the scaling policy to the registered variant. To apply a scaling policy to a variant, you can use the AWS CLI or the Application Auto Scaling API.

Apply a Scaling Policy to a Production Variant (AWS CLI)

To apply a scaling policy to your variant, use the `put-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--policy-type`—Set this value to `TargetTrackingScaling`.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--target-tracking-scaling-policy-configuration`—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses the `application-autoscaling put-scaling-policy` command with the following parameters:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id endpoint/MyEndpoint/variant/MyVariant \
    --service-namespace sagemaker \
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Apply a Scaling Policy to a Production Variant (Application Auto Scaling API)

To apply a scaling policy to a variant with the Application Auto Scaling API, use the `PutScalingPolicy Application Auto Scaling API` action with the following parameters:
Edit a Scaling Policy

You can edit a variant scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Edit a Scaling Policy (Console)

To edit a scaling policy with the AWS Management Console, use the same procedure that you used to Configure Automatic Scaling for a Production Variant (Console) (p. 697).

Edit a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to sagemaker.
- **ResourceId**—The resource identifier for the variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example, endpoint/MyEndpoint/variant/MyVariant.
- **ScalableDimension**—Set this value to sagemaker:variant:DesiredInstanceCount.
- **PolicyType**—Set this value to TargetTrackingScaling.
- **TargetTrackingScalingPolicyConfiguration**—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses Application Auto Scaling to apply a target-tracking scaling policy named myscalablepolicy to a variant named myscalablevariant. It uses a policy configuration based on the SageMakerVariantInvocationsPerInstance predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 70.0,
        "PredefinedMetricSpecification": {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
        }
    }
}
```
With the AWS CLI, specify the name of the policy that you want to edit in the `--policy-name` parameter. Specify new values for the parameters that you want to change.

With the Application Auto Scaling API, specify the name of the policy that you want to edit in the `PolicyName` parameter. Specify new values for the parameters that you want to change.

For more information, see Apply a Scaling Policy to a Production Variant (p. 702).

Delete a Scaling Policy

You can delete a scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Delete a Scaling Policy (Console)

To delete an automatic scaling policy for a variant (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint for which you want to delete automatic scaling.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. Choose **Configure auto scaling**.
6. Choose **Deregister auto scaling**.

Delete a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from a variant.

Delete a Scaling Policy (AWS CLI)

To delete a scaling policy from a variant, use the `delete-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

Example

The following example deletes a target-tracking scaling policy named `myscalablepolicy` from a variant named `myscalablevariant`.

```bash
aws application-autoscaling delete-scaling-policy \
  --policy-name myscalablepolicy \
  --resource-id endpoint/MyEndpoint/variant/MyVariant \
  --service-namespace sagemaker
```
Delete a Scaling Policy (Application Auto Scaling API)

To delete a scaling policy from your variant, use the DeleteScalingPolicy Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to sagemaker.
- **ResourceId**—The resource identifier for the variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example, endpoint/MyEndpoint/variant/MyVariant.
- **ScalableDimension**—Set this value to sagemaker:variant:DesiredInstanceCount.

Example

The following example uses the Application Auto Scaling API to delete a target-tracking scaling policy named myscalablepolicy from a variant named myscalablevariant.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
   "PolicyName": "myscalablepolicy",
   "ServiceNamespace": "sagemaker",
   "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
   "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```

Update Endpoints that Use Automatic Scaling

When you update Amazon SageMaker endpoints that have automatic scaling applied, complete the following steps:

To update an endpoint that has automatic scaling applied

1. Deregister the endpoint as a scalable target by calling DeregisterScalableTarget.
2. Because you turn off automatic scaling before you update the endpoint, you might want to take the additional precaution of increasing the number of instances for your endpoint during the update. To do this, update the instance counts for the production variants hosted at the endpoint by calling UpdateEndpointWeightsAndCapacities (p. 1235).
3. Call DescribeEndpoint (p. 1012) repeatedly until the value of the EndpointStatus field of the response is InService.
4. Call DescribeEndpointConfig (p. 1015) to get the values of the current endpoint config.
5. Create a new endpoint config by calling CreateEndpointConfig (p. 878). For the InitialInstanceCount field of each production variant, specify the corresponding value of
**DesiredInstanceCount** from the response to the previous call to `DescribeEndpoint (p. 1012)`. For all other values, use the values that you got as the response when you called `DescribeEndpointConfig (p. 1015)` in the previous step.

6. Update the endpoint by calling `UpdateEndpoint (p. 1233)`. Specify the endpoint config you created in the previous step as the `EndpointConfig` field.

7. Re-enable automatic scaling by calling `RegisterScalableTarget`.

**Load Testing for Production Variant Automatic Scaling**

Perform load tests to choose an automatic scaling configuration that works the way you want.

For an example of load testing to optimize automatic scaling for an Amazon SageMaker endpoint, see Load test and optimize an Amazon SageMaker endpoint using automatic scaling.

The following guidelines for load testing assume you are using an automatic scaling policy that uses the predefined target metric `SageMakerVariantInvocationsPerInstance`.

**Topics**
- **Determine the Performance Characteristics of a Production Variant (p. 706)**
- **Calculate the Target SageMakerVariantInvocationsPerInstance (p. 706)**

**Determine the Performance Characteristics of a Production Variant**

Perform load testing to find the peak `InvocationsPerInstance` that your variant instance can handle, and the latency of requests, as concurrency increases.

This value depends on the instance type chosen, payloads that clients of your variant typically send, and the performance of any external dependencies your variant has.

**To find the peak requests-per-second (RPS) your variant can handle and latency of requests**

1. Set up an endpoint with your variant using a single instance. For information about how to set up an endpoint, see Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 35).

2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the output of the load testing tool.

   **Note**
   You can also monitor requests-per-minute instead of RPS. In that case don’t multiply by 60 in the equation to calculate `SageMakerVariantInvocationsPerInstance` shown below.

   When the model latency increases or the proportion of successful transactions decreases, this is the peak RPS that your variant can handle.

**Calculate the Target SageMakerVariantInvocationsPerInstance**

After you find the performance characteristics of the variant, you can determine the maximum RPS we should allow to be sent to an instance. The threshold used for scaling must be less than this maximum value. Use the following equation in combination with load testing to determine the correct value for the `SageMakerVariantInvocationsPerInstance` target metric in your automatic scaling configuration.
SageMakerVariantInvocationsPerInstance = (MAX_RPS * SAFETY_FACTOR) * 60

Where MAX_RPS is the maximum RPS that you determined previously, and SAFETY_FACTOR is the safety factor that you chose to ensure that your clients don't exceed the maximum RPS. Multiply by 60 to convert from RPS to invocations-per-minute to match the per-minute CloudWatch metric that Amazon SageMaker uses to implement automatic scaling (you don't need to do this if you measured requests-per-minute instead of requests-per-second).

**Note**
Amazon SageMaker recommends that you start testing with a SAFETY_FACTOR of 0.5. Test your automatic scaling configuration to ensure it operates in the way you expect with your model for both increasing and decreasing customer traffic on your endpoint.

**Best Practices for Configuring Automatic Scaling**

When configuring automatic scaling, consider the following general guidelines.

**Testing Your Automatic Scaling Configuration**

It is important that you test your automatic scaling configuration to confirm that it works with your model the way you expect it to.

**Updating Endpoints Configured for Automatic Scaling**

When you update an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If the update would change the instance type for any variant that is a target for automatic scaling, the update fails.

In the AWS Management Console, you see a warning that you must deregister the variant from automatic scaling before you can update it. If you are trying to update the endpoint by calling the UpdateEndpoint (p. 1233) API, the call fails. Before you update the endpoint, delete any scaling policies configured for it by calling the DeleteScalingPolicy Application Auto Scaling API action, then call DeregisterScalableTarget to deregister the variant as a scalable target. After you update the endpoint, you can register the variant as a scalable target and attach an automatic scaling policy to the updated variant.

There is one exception. If you change the model for a variant that is configured for automatic scaling, Amazon SageMaker automatic scaling allows the update. This is because changing the model doesn't typically affect performance enough to change automatic scaling behavior. If you do update a model for a variant configured for automatic scaling, ensure that the change to the model doesn't significantly affect performance and automatic scaling behavior.

For instructions on how to update an endpoint that uses automatic scaling, see Update Endpoints that Use Automatic Scaling (p. 705).

**Deleting Endpoints Configured for Automatic Scaling**

If you delete an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If any are and you have permission to deregister the variant, Application Auto Scaling deregisters those variants as scalable targets without notifying you. If you use a custom permission policy that doesn't provide permission for the DeleteScalingPolicy and DeregisterScalableTarget actions, you must delete automatic scaling policies and deregister scalable targets and before deleting the endpoint.

**Note**
You, as an IAM user, might not have sufficient permission to delete an endpoint if another IAM user configured automatic scaling for a variant on that endpoint.
Using Step Scaling Policies

Although Amazon SageMaker automatic scaling supports using Application Auto Scaling step scaling policies, we recommend using target tracking policies, instead. For information about using Application Auto Scaling step scaling policies, see Step Scaling Policies.

Scaling In When There Is No Traffic

If a variant's traffic becomes zero, Amazon SageMaker automatic scaling doesn't scale down. This is because Amazon SageMaker doesn't emit metrics with a value of zero.

As a workaround, do either of the following:

• Send requests to the variant until automatic scaling scales down to the minimum capacity
• Change the policy to reduce the maximum provisioned capacity to match the minimum provisioned capacity

Troubleshoot Amazon SageMaker Model Deployments

If you encounter an issue when deploying machine learning models in Amazon SageMaker, see the following guidance.

Topics

• Detection Errors in the Active CPU Count (p. 708)

Detection Errors in the Active CPU Count

If you deploy an Amazon SageMaker model with a Linux Java Virtual Machine (JVM), you might encounter detection errors that prevent using available CPU resources. This issue affects some JVMs that support Java 8 and Java 9, and most that support Java 10 and Java 11. These JVMs implement a mechanism that detects and handles the CPU count and the maximum memory available when running a model in a Docker container, and, more generally, within Linux taskset commands or control groups (cgroups). Amazon SageMaker deployments take advantage of some of the settings that the JVM uses for managing these resources. Currently, this causes the container to incorrectly detect the number of available CPUs.

Amazon SageMaker doesn't limit access to CPUs on an instance. However, the JVM might detect the CPU count as 1 when more CPUs are available for the container. As a result, the JVM adjusts all of its internal settings to run as if only 1 CPU core is available. These settings affect garbage collection, locks, compiler threads, and other JVM internals that negatively affect the concurrency, throughput, and latency of the container.

For an example of the misdetection, in a container configured for Amazon SageMaker that is deployed with a JVM that is based on Java8_191 and that has four available CPUs on the instance, run the following command to start your JVM:

```bash
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_process_count: sched_getaffinity processor count: 4
```
Many of the JVMs affected by this issue have an option to disable this behavior and reestablish full access to all of the CPUs on the instance. Disable the unwanted behaviour and establish full access to all instance CPUs by including the \texttt{-XX:-UseContainerSupport} parameter when starting Java applications. For example, run the \texttt{java} command to start your JVM as follows:

\begin{verbatim}
java -XX:-UseContainerSupport -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
\end{verbatim}

This generates the following output:

\begin{verbatim}
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: sched_getaffinity processor count: 4
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
\end{verbatim}

Check whether the JVM used in your container supports the \texttt{-XX:-UseContainerSupport} parameter. If it does, always pass the parameter when you start your JVM. This provides access to all of the CPUs in your instances.

You might also encounter this issue when indirectly using a JVM in Amazon SageMaker containers. For example, when using a JVM to support SparkML Scala. The \texttt{-XX:-UseContainerSupport} parameter also affects the output returned by the \texttt{Java Runtime.getRuntime().availableProcessors()} API.

### Deployment Best Practices

This topic provides guidance on best practices for deploying machine learning models in Amazon SageMaker.

**Topics**

- Deploy Multiple Instances Across Availability Zones (p. 709)

### Deploy Multiple Instances Across Availability Zones

**Create robust endpoints when hosting your model.** Amazon SageMaker endpoints can help protect your application from Availability Zone outages and instance failures. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones. For this reason, we strongly recommended that you deploy multiple instances for each production endpoint.

If you are using an Amazon Virtual Private Cloud (VPC), configure the VPC with at least two Subnets, each in a different Availability Zone. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones.
In general, to achieve more reliable performance, use more small Instance Types in different Availability Zones to host your endpoints.

Host Instance Storage Volumes

When you create an endpoint, Amazon SageMaker attaches an Amazon Elastic Block Store (Amazon EBS) storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type.

The following table shows the size of the storage volume that Amazon SageMaker attaches for each instance type for a single endpoint and for a multi-model endpoint. For a MultiModel-enabled container, the storage volume provisioned for its instances has more memory. This allows more models to be cached on the instance storage volume. MultiModel containers with GPU instance types (for example, P2, P3, and G4 instance families) aren't supported on multi-model endpoints.

**Note**

Because d instance types come with an NVMe SSD storage, Amazon SageMaker doesn't attach an Amazon EBS storage volume to these ML compute instances that host the multi-model endpoint.

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Storage Volume for Single Endpoint in GB</th>
<th>Storage Volume for Multi-Model Endpoint in GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>30</td>
<td>128</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>30</td>
<td>320</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>30</td>
<td>512</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>30</td>
<td>128</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>30</td>
<td>384</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>30</td>
<td>768</td>
</tr>
<tr>
<td>Instance Type</td>
<td>Storage Volume in GB</td>
<td>Storage Volume for Multi-Model Endpoint in GB</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>30</td>
<td>120</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>16</td>
<td>72</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>30</td>
<td>144</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>30</td>
<td>288</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>30</td>
<td>Not supported</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>30</td>
<td>128</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>30</td>
<td>256</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>30</td>
<td>762</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>30</td>
<td>1536</td>
</tr>
</tbody>
</table>
Monitor Amazon SageMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SageMaker and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon SageMaker, report when something is wrong, and take automatic actions when appropriate:

- **Amazon CloudWatch** monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.

- **Amazon CloudWatch Logs** enables you to monitor, store, and access your log files from EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

- **CloudWatch Events** delivers a near real-time stream of system events that describe changes in AWS resources. Create CloudWatch Events rules react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job.

Topics
- Monitor Amazon SageMaker with Amazon CloudWatch (p. 712)
- Log Amazon SageMaker Events with Amazon CloudWatch (p. 719)
- Log Amazon SageMaker API Calls with AWS CloudTrail (p. 720)
- React to Amazon SageMaker Job Status Changes with CloudWatch Events (p. 723)

Monitor Amazon SageMaker with Amazon CloudWatch

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. However, the Amazon CloudWatch console limits the search to metrics that were updated in the last 2 weeks. This limitation ensures that the most current jobs are shown in your namespace. To graph metrics without using a search, specify its exact name in the source view. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

Amazon SageMaker model training jobs and endpoints write CloudWatch metrics and logs. The following tables list the metrics and dimensions for Amazon SageMaker.

**Endpoint Invocation Metrics**
The AWS/SageMaker namespace includes the following request metrics from calls to `InvokeEndpoint` (p. 1260).

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see `GetMetricStatistics` in the `Amazon CloudWatch API Reference`.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation4XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocation5XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocations</td>
<td>The number of <code>InvokeEndpoint</code> requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>InvocationsPerInstance</td>
<td>The number of invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code>. $1/\text{numberOfInstances}$ is sent as the value on each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> behind the endpoint at the time of the request. Units: None Valid statistics: Sum</td>
</tr>
<tr>
<td>ModelLatency</td>
<td>The interval of time taken by a model to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>OverheadLatency</td>
<td>The interval of time added to the time taken to respond to a client request by Amazon SageMaker overheads. This interval is measured from the time Amazon SageMaker receives the request until it returns a response to the client, minus the ModelLatency. Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request. Units: Microseconds</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
</tbody>
</table>

**Dimensions for Endpoint Invocation Metrics**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndpointName, VariantName</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
</tbody>
</table>

**Multi-Model Endpoint Model Loading Metrics**

The AWS/SageMaker namespace includes the following model loading metrics from calls to `InvokeEndpoint` (p. 1260).

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see `GetMetricStatistics` in the Amazon CloudWatch API Reference.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelLoadingWaitTime</td>
<td>The interval of time that an invocation request has waited for the target model to be downloaded, or loaded, or both in order to perform inference. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>ModelUnloadingTime</td>
<td>The interval of time that it took to unload the model through the container's <code>UnloadModel</code> API call. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>ModelDownloadingTime</td>
<td>The interval of time that it took to download the model from Amazon Simple Storage Service (Amazon S3). Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>ModelLoadingTime</td>
<td>The interval of time that it took to load the model through the container's <code>LoadModel</code> API call. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>ModelCacheHit</td>
<td>The number of <code>InvokeEndpoint</code> requests sent to the multi-model endpoint for which the model was already loaded. The Average statistic shows the ratio of requests for which the model was already loaded. Units: None</td>
</tr>
</tbody>
</table>
## Metric Description

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid statistics: Average, Sum, Sample Count</td>
</tr>
</tbody>
</table>

### Dimensions for Multi-Model Endpoint Model Loading Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndpointName, VariantName</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
</tbody>
</table>

## Multi-Model Endpoint Model Instance Metrics

The `/AWS/SageMaker/Endpoints` namespaces include the following instance metrics from calls to `InvokeEndpoint (p. 1260)`.

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see `GetMetricStatistics` in the *Amazon CloudWatch API Reference*.

### Metric Description

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadedModelCount</td>
<td>The number of models loaded in the containers of the multi-model endpoint. This metric is emitted per instance.</td>
</tr>
<tr>
<td></td>
<td>The Average statistic with a period of 1 minute tells you the average number of models loaded per instance.</td>
</tr>
<tr>
<td></td>
<td>The Sum statistic tells you the total number of models loaded across all instances in the endpoint.</td>
</tr>
<tr>
<td></td>
<td>The models that this metric tracks are not necessarily unique because a model might be loaded in multiple containers at the endpoint.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
</tbody>
</table>

### Dimensions for Multi-Model Endpoint Model Loading Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndpointName, VariantName</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
</tbody>
</table>

## Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics

The `/aws/sagemaker/ProcessingJobs`, `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs` and `/aws/sagemaker/Endpoints` namespaces include the following metrics for the training jobs and endpoint instances.

Metrics are available at a 1-minute frequency.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUtilization</td>
<td>The percentage of CPU units that are used by the containers on an instance. The value can range between 0 and 100, and is multiplied by the number of CPUs. For example, if there are four CPUs, CPUUtilization can range from 0% to 400%. For processing jobs, the value is the CPU utilization of the processing container on the instance. For training jobs, the value is the CPU utilization of the algorithm container on the instance. For batch transform jobs, the value is the CPU utilization of the transform container on the instance. For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance. Note For multi-instance, each instance reports CPU utilization metrics. However, the default view in CloudWatch shows the average CPU utilization across all instances. Units: Percent</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%. For processing jobs, the value is the memory utilization of the processing container on the instance. For training jobs, the value is the memory utilization of the algorithm container on the instance. For batch transform jobs, the value is the memory utilization of the transform container on the instance. For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance. Note For multi-instance, each instance reports memory utilization metrics. However, the default view in CloudWatch shows the average memory utilization across all instances. Units: Percent</td>
</tr>
<tr>
<td>GPUUtilization</td>
<td>The percentage of GPU units that are used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUUtilization can range from 0% to 400%. For processing jobs, the value is the GPU utilization of the processing container on the instance. For training jobs, the value is the GPU utilization of the algorithm container on the instance. For batch transform jobs, the value is the GPU utilization of the transform container on the instance.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>For multi-instance, each instance reports GPU utilization metrics. However, the default view in CloudWatch shows the average GPU utilization across all instances.</td>
</tr>
<tr>
<td><strong>Units</strong>: Percent</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>GPUMemoryUtilization</strong></td>
</tr>
<tr>
<td></td>
<td><strong>DiskUtilization</strong></td>
</tr>
</tbody>
</table>

**Dimensions for Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics**
<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
</table>
| Host       | For processing jobs, the value for this dimension has the format [processing-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified processing job and instance. This dimension format is present only in the /aws/sagemaker/ProcessingJobs namespace.  
For training jobs, the value for this dimension has the format [training-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the /aws/sagemaker/TrainingJobs namespace.  
For batch transform jobs, the value for this dimension has the format [transform-job-name]/[instance-id]. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the /aws/sagemaker/TransformJobs namespace.  
For endpoints, the value for this dimension has the format [endpoint-name]/[production-variant-name ]/[instance-id]. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the /aws/sagemaker/Endpoints namespace. |

**Amazon SageMaker Ground Truth Metrics**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
</table>
| ActiveWorkers           | The number of workers on a private work team performing a labeling job.  
Units: None  
Valid statistics: Max |
| DatasetObjectsAutoAnnotated | The number of dataset objects auto-annotated in a labeling job. This metric is only emitted when automated labeling is enabled. To view the labeling job progress, use the Max metric.  
Units: None  
Valid statistics: Max |
| DatasetObjectsHumanAnnotated | The number of dataset objects annotated by a human in a labeling job. To view the labeling job progress, use the Max metric.  
Units: None  
Valid statistics: Max |
| DatasetObjectsLabelingFailed | The number of dataset objects that failed labeling in a labeling job. To view the labeling job progress, use the Max metric.  
Units: None  
Valid statistics: Max |
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobsFailed</td>
<td>The number of labeling jobs that failed. To get the total number of labeling jobs that failed, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>JobsSucceeded</td>
<td>The number of labeling jobs that succeeded. To get the total number of labeling jobs that succeeded, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>JobsStopped</td>
<td>The number of labeling jobs that were stopped. To get the total number of labeling jobs that were stopped, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>TasksSubmitted</td>
<td>The number of tasks submitted/completed by a private work team.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Max</td>
</tr>
<tr>
<td>TimeSpent</td>
<td>Time spent on a task completed by a private work team.</td>
</tr>
<tr>
<td></td>
<td>Units: Seconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Max</td>
</tr>
<tr>
<td>TotalDatasetObjectsLabeled</td>
<td>The number of dataset objects labeled successfully in a labeling job. To view the labeling job progress, use the Max metric.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Max</td>
</tr>
</tbody>
</table>

**Dimensions for Dataset Object Metrics**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LabelingJobName</td>
<td>Filters dataset object count metrics for a labeling job.</td>
</tr>
</tbody>
</table>

Log Amazon SageMaker Events with Amazon CloudWatch

To help you debug your processing jobs, training jobs, endpoints, transform jobs, notebook instances, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.
Log Amazon SageMaker API Calls with AWS CloudTrail

Amazon SageMaker is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. CloudTrail captures all API calls for Amazon SageMaker, with the exception of `InvokeEndpoint` (p. 1260), as events. The calls captured include:

The following table lists all of the logs provided by Amazon SageMaker.

### Logs

<table>
<thead>
<tr>
<th>Log Group Name</th>
<th>Log Stream Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aws/sagemaker/ProcessingJobs</td>
<td>[processing-job-name]/[hostname]-[epoch_timestamp]</td>
</tr>
<tr>
<td>/aws/sagemaker/TrainingJobs</td>
<td>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</td>
</tr>
<tr>
<td>/aws/sagemaker/Endpoints/[EndpointName]</td>
<td>[production-variant-name]/[instance-id]</td>
</tr>
<tr>
<td>/aws/sagemaker/NotebookInstances</td>
<td>[production-variant-name]/[instance-id]/[container-name provided in SageMaker model] (For Inference Pipelines)</td>
</tr>
<tr>
<td>/aws/sagemaker/TransformJobs</td>
<td>[notebook-instance-name]/[LifecycleConfigHook]</td>
</tr>
<tr>
<td>/aws/sagemaker/TransformJobs</td>
<td>[notebook-instance-name]/jupyter.log</td>
</tr>
<tr>
<td>/aws/sagemaker/LabelingJobs</td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]</td>
</tr>
<tr>
<td>/aws/sagemaker/LabelingJobs</td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]/datalog</td>
</tr>
<tr>
<td>/aws/sagemaker/groundtruth/WorkerActivity</td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]/[container-name provided in SageMaker model] (For Inference Pipelines)</td>
</tr>
<tr>
<td>/aws/sagemaker/LabelingJobs</td>
<td>[labeling-job-name]</td>
</tr>
<tr>
<td>/aws/sagemaker/groundtruth/WorkerActivity</td>
<td>aws/sagemaker/groundtruth/worker-activity/[requester-AWS-Id]-[region]/[timestamp]</td>
</tr>
</tbody>
</table>

**Note**

1. The `/aws/sagemaker/NotebookInstances/[LifecycleConfigHook]` log stream is created when you create a notebook instance with a lifecycle configuration. For more information, see Customize a Notebook Instance (p. 206).
2. For Inference Pipelines, if you don't provide container names, the platform uses **container-1, container-2**, and so on, corresponding to the order provided in the Amazon SageMaker model.

For more information about logging events with CloudWatch logging, see What is Amazon CloudWatch Logs? in the Amazon CloudWatch User Guide.
calls from the Amazon SageMaker console and code calls to the Amazon SageMaker API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon SageMaker. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

By default, log data is stored in CloudWatch Logs indefinitely. However, you can configure how long to store log data in a log group. For information, see Change Log Data Retention in CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

Amazon SageMaker Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon SageMaker, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon SageMaker, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon SageMaker actions, with the exception of InvokeEndpoint (p. 1260), are logged by CloudTrail and are documented in the Actions (p. 843). For example, calls to the CreateTrainingJob, CreateEndpoint and CreateNotebookInstance actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Operations Performed by Automatic Model Tuning

Amazon SageMaker supports logging non-API service events to your CloudTrail log files, for automatic model tuning jobs. These events are related to your tuning jobs but, are not the direct result of a customer request to the public AWS API. For example, when you create a hyperparameter tuning job by calling CreateHyperParameterTuningJob (p. 890), Amazon SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call
StopHyperParameterTuningJob (p. 1214) to stop a hyperparameter tuning job, Amazon SageMaker might stop any of the associated running training jobs. Non-API events for your tuning jobs are logged to CloudTrail to help you improve governance, compliance, and operational and risk auditing of your AWS account.

Log entries that result from non-API service events have an eventType of AwsServiceEvent instead of AwsApiCall.

Understanding Amazon SageMaker Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following examples a log entry for the CreateEndpoint action, which creates an endpoint to deploy a trained model.

```json
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIXDAYQEXAMPLEUMLYNGL",
    "arn": "arn:aws:iam::123456789012:user/intern",
    "accountId": "123456789012",
    "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
    "userName": "intern"
  },
  "eventTime": "2018-01-02T13:39:06Z",
  "eventSource": "sagemaker.amazonaws.com",
  "eventName": "CreateEndpoint",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "USER_AGENT",
  "requestParameters": {
    "endpointName": "ExampleEndpoint",
    "endpointConfigName": "ExampleEndpointConfig"
  },
  "responseElements": {
    "endpointArn": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/exampleendpoint"
  },
  "requestID": "6b1b42b9-EXAMPLE",
  "eventID": "a6f85b21-EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "444455556666"
}
```

The following example is a log entry for the CreateModel action, which creates one or more containers to host a previously trained model.

```json
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIXDAYQEXAMPLEUMLYNGL",
    "arn": "arn:aws:iam::123456789012:user/intern",
    "accountId": "123456789012",
    "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
    "userName": "intern"
  },
  "eventTime": "2018-01-02T13:39:06Z",
  "eventSource": "sagemaker.amazonaws.com",
  "eventName": "CreateModel",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "USER_AGENT",
  "requestParameters": {
    "modelName": "ExampleModel",
    "containerImage": "exampleimage:1.0",
    "containerConfig": "ExampleConfig"
  },
  "responseElements": {
    "modelArn": "arn:aws:sagemaker:us-west-2:123456789012:model/examplemodel"
  },
  "requestID": "6b1b42b9-EXAMPLE",
  "eventID": "a6f85b21-EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "444455556666"
}
```
React to Amazon SageMaker Job Status Changes with CloudWatch Events

To react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job, create a rule in CloudWatch Events that use the SageMaker Training Job State Change, SageMaker Hyperparameter Tuning Job State Change, or SageMaker Transform Job State Change event type as the event source for the rule.

Every time the status of a Amazon SageMaker job changes, it triggers an event that CloudWatch Events monitors, and you can create a rule that calls a AWS Lambda function when the status changes. For information about the status values and meanings for Amazon SageMaker jobs, see the following:

- TrainingJobStatus
- HyperParameterTuningJobStatus
- TransformJobStatus

For information about creating CloudWatch Events rules, see Creating a CloudWatch Events Rule That Triggers on an Event in the CloudWatch Events User Guide. For detailed information about the format of the Amazon SageMaker events that CloudWatch Events monitors, see Amazon SageMaker Events.
Security in Amazon SageMaker

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon SageMaker, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon SageMaker. The following topics show you how to configure Amazon SageMaker to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon SageMaker resources.

**Topics**

- Data Protection in Amazon SageMaker (p. 724)
- Identity and Access Management for Amazon SageMaker (p. 728)
- Logging and Monitoring (p. 778)
- Compliance Validation for Amazon SageMaker (p. 778)
- Resilience in Amazon SageMaker (p. 779)
- Infrastructure Security in Amazon SageMaker (p. 779)

Data Protection in Amazon SageMaker

Amazon SageMaker conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a Name field. This includes when you work with Amazon SageMaker or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon SageMaker or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don’t include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

Topics
• Protecting Data at Rest Using Encryption (p. 725)
• Protecting Data in Transit with Encryption (p. 726)
• Key Management (p. 728)
• Internetwork Traffic Privacy (p. 728)

Protecting Data at Rest Using Encryption

You can use encrypted Amazon Simple Storage Service buckets for model artifacts and data, as well as pass an AWS Key Management Service key to Amazon SageMaker notebooks, processing jobs, training jobs, hyperparameter tuning jobs, batch transform jobs, and endpoints, to encrypt the attached machine learning (ML) storage volume. If you do not specify an AWS KMS key, Amazon SageMaker encrypts storage volumes with a transient key. A transient key is discarded immediately after it is used to encrypt the storage volume.

All instance OS volumes are encrypted with an AWS-managed AWS KMS key.

All ML data volumes for all Amazon SageMaker instances may be encrypted with customer specified AWS KMS keys. ML data volumes are mounted as follows:

• Notebooks - /home/ec2-user/SageMaker
• Processing - /opt/ml/processing and /tmp/
• Training - /opt/ml/ and /tmp/
• Batch - /opt/ml/ and /tmp/
• Endpoints - /opt/ml/ and /tmp/

Processing, batch and training job containers and their storage are ephemeral in nature. When the job completes, output is uploaded to Amazon S3 (with optional AWS KMS encryption) and the instance is torn down.

Data of a sensitive nature that needs to be encrypted with a customer owned AWS KMS key for compliance reasons should be stored in the ML Amazon EBS volume or Amazon S3, both of which can be KMS encrypted with customer managed keys. Notebook instances mount all default folders used by Jupyter or the algorithm containers onto the ML volume.

The Amazon SageMaker folder in the ML Amazon EBS volume is the default storage location when you open a notebook instance. Amazon SageMaker saves any files within the SageMaker folder. The /sample-notebooks subfolder is located on the OS volume but that location is read only. When you stop a Notebook instance any customizations to the OS (like custom libraries installed or OS level settings) are lost. Consider utilizing lifecycle options to automate any customizations to the default image. If a Notebook instance is stopped, a snapshot of the ML volume is retained by Amazon in the service platform to support resumption. This snapshot is deleted on termination as well as the ML volume, so any data to be persisted beyond the notebook lifecycle should be transferred to customer Amazon S3 buckets.
Note
Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a VolumeKmsKeyId when using an instance type with local storage.
For a list of instance types that support local instance storage, see Instance Store Volumes. For more information about local instance storage encryption, see SSD Instance Store Volumes. For more information about storage volumes on nitro-based instances, see Amazon EBS and NVMe on Linux Instances.

Protecting Data in Transit with Encryption

All inter-network data in transit supports TLS 1.2 encryption.

Amazon SageMaker ensures that machine learning (ML) model artifacts and other system artifacts are encrypted in transit and at rest. Requests to the Amazon SageMaker API and console are made over a secure (SSL) connection. You pass AWS Identity and Access Management roles to Amazon SageMaker to provide permissions to access resources on your behalf for training and deployment. You can use encrypted Amazon S3 buckets for model artifacts and data, as well as pass a AWS KMS key to Amazon SageMaker instances to encrypt the attached ML storage volumes.

Some intra-network data in-transit (inside the service platform) is unencrypted. This includes:

- Command and control communications between the service control plane and training job instances (not customer data).
- Communications between nodes in distributed processing jobs (intra-network).
- Communications between nodes in distributed training jobs (intra-network).

There are no inter-node communications for batch processing.

You can choose to encrypt internode training communications. Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

FIPS validated endpoints are available for the Amazon SageMaker API and request router for hosted models (runtime). For information about FIPS compliant endpoints, see Federal Information Processing Standard (FIPS) 140-2.

Protect Communications Between ML Compute Instances in a Distributed Training Job

By default, Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud (Amazon VPC) to help keep your data secure. You can add another level of security to protect your training containers and data by configuring a private VPC. Distributed ML frameworks and algorithms usually transmit information that is directly related to the model such as weights, not the training dataset. When performing distributed training, you can further protect data that is transmitted between instances. This can help you to comply with regulatory requirements. To do this, use inter-container traffic encryption.

Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. Enabling inter-container traffic encryption doesn't affect training jobs with a single compute instance. However, for training jobs with several compute instances, the effect on training time depends on the amount of communication between compute instances. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.
You can enable inter-container traffic encryption for training jobs or hyperparameter tuning jobs. You can use Amazon SageMaker APIs or console to enable inter-container traffic encryption.

For information about running training jobs in a private VPC, see Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 789).

**Enable Inter-Container Traffic Encryption (API)**

Before enabling inter-container traffic encryption on training or hyperparameter tuning jobs with APIs, you need to add inbound and outbound rules to your private VPC's security group.

**To enable inter-container traffic encryption (API)**

1. Add the following inbound and outbound rules in the security group for your private VPC:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>500</td>
<td><em>Self Security Group ID</em></td>
</tr>
<tr>
<td>50</td>
<td>N/A</td>
<td><em>Self Security Group ID</em></td>
</tr>
</tbody>
</table>

2. When you send a request to the CreateTrainingJob (p. 931) or CreateHyperParameterTuningJob (p. 890) API, specify True for the EnableInterContainerTrafficEncryption parameter.

**Note**

The AWS Security Group Console might show display ports range as "All", however EC2 ignores the specified port range because it is not applicable for the ESP 50 IP protocol.

**Enable Inter-Container Traffic Encryption (Console)**

**Enable Inter-container Traffic Encryption in a Training Job**

**To enable inter-container traffic encryption in a training job**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker
2. In the navigation pane, choose Training, then choose Training jobs.
3. Choose Create training job.
4. Under Network, choose a VPC. You can use the default VPC or one that you have created.
5. Choose Enable inter-container traffic encryption.

After you enable inter-container traffic encryption, finish creating the training job. For more information, see Step 5: Train a Model (p. 30).

**Enable Inter-container Traffic Encryption in a Hyperparameter Tuning Job**

**To enable inter-container traffic encryption in a hyperparameter tuning job**

2. In the navigation pane, choose Training, then choose Hyperparameter tuning jobs.
3. Choose Create hyperparameter tuning job.
4. Under Network, choose a VPC. You can use the default VPC or one that you created.
5. Choose Enable inter-container traffic encryption.
After enabling inter-container traffic encryption, finish creating the hyperparameter tuning job. For more information, see Configure and Launch a Hyperparameter Tuning Job (p. 563).

Key Management

Customers can specify AWS KMS keys, including bring your own keys (BYOK), to use for envelope encryption with Amazon S3 input/output buckets and machine learning (ML) Amazon EBS volumes. ML volumes for notebook instances and for processing, training, and hosted model Docker containers can be optionally encrypted by using AWS KMS customer-owned keys. All instance OS volumes are encrypted with an AWS-managed AWS KMS key.

**Note**

Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a `VolumeKmsKeyId` when using an instance type with local storage.

For a list of instance types that support local instance storage, see Instance Store Volumes. For more information about local instance storage encryption, see SSD Instance Store Volumes. For more information about storage volumes on nitro-based instances, see Amazon EBS and NVMe on Linux Instances.

For information about AWS KMS keys see What is AWS Key Management Service? in the AWS Key Management Service Developer Guide.

Internetwork Traffic Privacy

This topic describes how Amazon SageMaker secures connections from the service to other locations.

Internetwork communications support TLS 1.2 encryption between all components and clients.

Instances can be connected to Customer VPC, providing access to S3 VPC endpoints or customer repositories. Internet egress can be managed through this interface by the customer if service platform internet egress is disabled for notebooks. For training and hosting, egress through the service platform is not available when connected to the customer's VPC.

By default, API calls made to published endpoints traverse the public network to the request router. Amazon SageMaker supports Amazon Virtual Private Cloud interface endpoints powered by AWS PrivateLink for private connectivity between the customer's VPC and the request router to access hosted model endpoints. For information about Amazon VPC, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 781)

Identity and Access Management for Amazon SageMaker

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use Amazon SageMaker resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience (p. 729)
- Authenticating with Identities (p. 729)
- Managing Access Using Policies (p. 731)
- How Amazon SageMaker Works with IAM (p. 732)
Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon SageMaker.

**Service user** – If you use the Amazon SageMaker service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon SageMaker features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon SageMaker, see Troubleshooting Amazon SageMaker Identity and Access (p. 776).

**Service administrator** – If you're in charge of Amazon SageMaker resources at your company, you probably have full access to Amazon SageMaker. It's your job to determine which Amazon SageMaker features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon SageMaker, see How Amazon SageMaker Works with IAM (p. 732).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon SageMaker. To view example Amazon SageMaker identity-based policies that you can use in IAM, see Amazon SageMaker Identity-Based Policy Examples (p. 735).

Authenticating with Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and
is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

**IAM Users and Groups**

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the IAM User Guide.

**IAM Roles**

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.
• **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the IAM User Guide.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the IAM User Guide.

### Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the IAM User Guide.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

#### Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the IAM User Guide.

Identity-based policies can be further categorized as **inline policies** or **managed policies**. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the IAM User Guide.

#### Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

#### Access Control Lists (ACLs)

Access control policies (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon...
VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the Amazon Simple Storage Service Developer Guide.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the IAM User Guide.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the IAM User Guide.

How Amazon SageMaker Works with IAM

Before you use IAM to manage access to Amazon SageMaker, you should understand what IAM features are available to use with Amazon SageMaker. To get a high-level view of how Amazon SageMaker and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

Topics

- Amazon SageMaker Identity-Based Policies (p. 732)

Amazon SageMaker Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon SageMaker supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.

Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.
Policy actions in Amazon SageMaker use the following prefix before the action: `sagemaker:`. For example, to grant someone permission to run an Amazon SageMaker training job with the Amazon SageMaker `CreateTrainingJob` API operation, you include the `sagemaker:CreateTrainingJob` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon SageMaker defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
    "sagemaker:action1",
    "sagemaker:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "sagemaker:Describe*"
```

To see a list of Amazon SageMaker actions, see Actions Defined by Amazon SageMaker in the IAM User Guide.

**Resources**

Amazon SageMaker does not support specifying resource ARNs in a policy.

**Condition Keys**

The `Condition` element (or `Condition block`) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the IAM User Guide.

Amazon SageMaker defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

Amazon SageMaker supports a number of service-specific condition keys that you can use for fine-grained access control for the following operations:

- the section called “CreateProcessingJob” (p. 926)
- the section called “CreateTrainingJob” (p. 931)
- the section called “CreateModel” (p. 902)
- the section called “CreateEndpointConfig” (p. 878)
- the section called “CreateTransformJob” (p. 939)
- the section called “CreateHyperParameterTuningJob” (p. 890)
- the section called “CreateLabelingJob” (p. 897)
- the section called “CreateNotebookInstance” (p. 913)
- the section called “UpdateNotebookInstance” (p. 1242)
To see a list of Amazon SageMaker condition keys, see Condition Keys for Amazon SageMaker in the IAM User Guide. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon SageMaker.

For examples of using Amazon SageMaker condition keys, see the following: Control Creation of Amazon SageMaker Resources with Condition Keys (p. 745).

Examples

To view examples of Amazon SageMaker identity-based policies, see Amazon SageMaker Identity-Based Policy Examples (p. 735).

Amazon SageMaker Resource-Based Policies

Amazon SageMaker does not support resource-based policies.

Authorization Based on Amazon SageMaker Tags

You can attach tags to Amazon SageMaker resources or pass tags in a request to Amazon SageMaker. To control access based on tags, you provide tag information in the condition element of a policy using the sagemaker:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. For more information about tagging Amazon SageMaker resources, see Control Access to Amazon SageMaker Resources by Using Tags (p. 754).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Control Access to Amazon SageMaker Resources by Using Tags (p. 754).

Amazon SageMaker IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon SageMaker

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon SageMaker supports using temporary credentials.

Service-Linked Roles

Amazon SageMaker doesn't support service-linked roles.

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon SageMaker supports service roles.

Choosing an IAM Role in Amazon SageMaker

When you create a notebook instance, processing job, training job, hosted endpoint, or batch transform job resource in Amazon SageMaker, you must choose a role to allow Amazon SageMaker to access Amazon SageMaker on your behalf. If you have previously created a service role or service-linked role, then Amazon SageMaker provides you with a list of roles to choose from. It's important to choose a role that allows access to the AWS operations and resources you need. For more information, see Amazon SageMaker Roles (p. 758).
Amazon SageMaker Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon SageMaker resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions. To learn how to attach policies to an IAM user or group, see Adding and Removing IAM Identity Permissions in the IAM User Guide.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.

Topics
- Policy Best Practices (p. 735)
- Using the Amazon SageMaker Console (p. 735)
- Allow Users to View Their Own Permissions (p. 744)
- Control Creation of Amazon SageMaker Resources with Condition Keys (p. 745)
- Control Access to the Amazon SageMaker API by Using Identity-based Policies (p. 752)
- Limit Access to Amazon SageMaker API and Runtime Calls by IP Address (p. 753)
- Limit Access to a Notebook Instance by IP Address (p. 754)
- Control Access to Amazon SageMaker Resources by Using Tags (p. 754)
- Require the Presence or Absence of Tags for API Calls (p. 756)
- Use Tags with Hyperparameter Tuning Jobs (p. 757)

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon SageMaker resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon SageMaker quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the IAM User Guide.

- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the IAM User Guide.

- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the IAM User Guide.

Using the Amazon SageMaker Console

To access the Amazon SageMaker console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon SageMaker resources in your
AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon SageMaker console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the IAM User Guide:

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Topics

- Permissions Required to Use the Amazon SageMaker Console (p. 736)
- Permissions Required to Use the Amazon SageMaker Ground Truth Console (p. 737)
- Permissions Required to Use the Amazon Augmented AI (Preview) Console (p. 738)

Permissions Required to Use the Amazon SageMaker Console

The permissions reference table lists the Amazon SageMaker API operations and shows the required permissions for each operation. For more information about Amazon SageMaker API operations, see Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 771).

To use the Amazon SageMaker console, you need to grant permissions for additional actions. Specifically, the console needs permissions that allow the ec2 actions to display subnets, VPCs, and security groups. Optionally, the console needs permission to create execution roles for tasks such as CreateNotebook, CreateTrainingJob, and CreateModel. Grant these permissions with the following permissions policy:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "SageMakerApis",
         "Effect": "Allow",
         "Action": ["sagemaker:*"],
         "Resource": "*"
      },
      {
         "Sid": "VpcConfigurationForCreateForms",
         "Effect": "Allow",
         "Action": ["ec2:DescribeVpcs",
                     "ec2:DescribeSubnets",
                     "ec2:DescribeSecurityGroups"],
         "Resource": "*"
      },
      {
         "Sid": "KmsKeysForCreateForms",
         "Effect": "Allow",
         "Action": ["kms:DescribeKey",
                     "kms:ListAliases"],
         "Resource": "*"
      },
      {
         "Sid": "AccessAwsMarketplaceSubscritions",
         "Effect": "Allow",
         "Action": ["aws-marketplace:*"],
         "Resource": "*"
      }
   ]
}
```
Permissions Required to Use the Amazon SageMaker Ground Truth Console

To use the Amazon SageMaker Ground Truth console, you need to grant permissions for additional resources. Specifically, the console needs permissions for the AWS Marketplace to view subscriptions, Amazon Cognito operations to manage your private workforce, Amazon S3 actions for access to your input and output files, and AWS Lambda actions to list and invoke functions. Grant these permissions with the following permissions policy:

```json
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:ViewSubscriptions"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGetRepositories",
                "codecommit:CreateRepository",
                "codecommit:GetRepository",
                "codecommit:ListRepositories",
                "codecommit:ListBranches",
                "secretsmanager:CreateSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListAndCreateExecutionRoles",
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DescribeECRMetaData",
            "Effect": "Allow",
            "Action": [
                "ecr:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassRoleForExecutionRoles",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        }
    ]
}
```
Permissions Required to Use the Amazon Augmented AI (Preview) Console

To use the Augmented AI console, you need to grant permissions for additional resources. Grant these permissions with the following permissions policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GroundTruthConsole",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:DescribeListings",
        "aws-marketplace:ViewSubscriptions",
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp:CreateGroup",
        "cognito-idp:CreateUserPool",
        "cognito-idp:CreateUserPoolClient",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp:DescribeUserPoolDomain",
        "cognito-idp:ListGroups",
        "cognito-idp:ListIdentityProviders",
        "cognito-idp:ListUsers",
        "cognito-idp:ListUsersInGroup",
        "cognito-idp:ListUserPoolClients",
        "cognito-idp:ListUserPools",
        "cognito-idp:UpdateUserPool",
        "cognito-idp:UpdateUserPoolClient",
        "groundtruthlabeling:DescribeConsoleJob",
        "groundtruthlabeling:ListDatasetObjects",
        "groundtruthlabeling:RunFilterOrSampleManifestJob",
        "groundtruthlabeling:RunGenerateManifestByCrawlingJob",
        "lambda:InvokeFunction",
        "lambda:ListFunctions",
        "s3:GetObject",
        "s3:PutObject",
        "s3:SelectObjectContent"
      ],
      "Resource": "*
    }
  ]
}
```
"sagemaker:*CodeRepositories",
"sagemaker:*CodeRepository",
"sagemaker:*CompilationJob",
"sagemaker:*CompilationJobs",
"sagemaker:*Endpoint",
"sagemaker:*EndpointConfig",
"sagemaker:*EndpointConfigs",
"sagemaker:*EndpointWeightsAndCapacities",
"sagemaker:*Endpoints",
"sagemaker:*Environment",
"sagemaker:*EnvironmentVersion",
"sagemaker:*EnvironmentVersions",
"sagemaker:*Environments",
"sagemaker:*Experiment",
"sagemaker:*Experiments",
"sagemaker:*FlowDefinitions",
"sagemaker:*HumanLoop",
"sagemaker:*HumanLoops",
"sagemaker:*HumanTaskUi",
"sagemaker:*HumanTaskUis",
"sagemaker:*HyperParameterTuningJob",
"sagemaker:*HyperParameterTuningJobs",
"sagemaker:*LabelingJob",
"sagemaker:*LabelingJobs",
"sagemaker:*Metrics",
"sagemaker:*Model",
"sagemaker:*ModelPackage",
"sagemaker:*ModelPackages",
"sagemaker:*Models",
"sagemaker:*MonitoringExecutions",
"sagemaker:*MonitoringSchedule",
"sagemaker:*MonitoringSchedules",
"sagemaker:*NotebookInstance",
"sagemaker:*NotebookInstanceLifecycleConfig",
"sagemaker:*NotebookInstanceLifecycleConfigs",
"sagemaker:*NotebookInstanceUrl",
"sagemaker:*NotebookInstances",
"sagemaker:*ProcessingJob",
"sagemaker:*ProcessingJobs",
"sagemaker:*RenderUiTemplate",
"sagemaker:*Search",
"sagemaker:*SearchSuggestions",
"sagemaker:*Tags",
"sagemaker:*TrainingJob",
"sagemaker:*TrainingJobs",
"sagemaker:*TransformJob",
"sagemaker:*TransformJobs",
"sagemaker:*Trial",
"sagemaker:*TrialComponent",
"sagemaker:*TrialComponents",
"sagemaker:*Trials",
"sagemaker:*Workteam",
"sagemaker:*Workteams"
],
"Resource": "*
",
{
"Effect": "Allow",
"Action": [
"sagemaker:*FlowDefinition"
],
"Resource": "*
",
"Condition": {
"StringEqualsIfExists": {
"sagemaker:WorkteamType": [
"private-crowd",
"private-crowd"
]}}}
}
"vendor-crowd"
"ecr:Describe*",
"ecr:GetAuthorizationToken",
"ecr:GetDownloadUrlForLayer",
"elastic-inference:Connect",
"elasticfilesystem:DescribeFileSystems",
"elasticfilesystem:DescribeMountTargets",
"fsx:DescribeFileSystems",
"glue:CreateJob",
"glue:DeleteJob",
"glue:GetJob",
"glue:GetJobRun",
"glue:GetJobRuns",
"glue:GetJobs",
"glue:ResetJobBookmark",
"glue:StartJobRun",
"glue:UpdateJob",
"groundtruthlabeling:**",
"iam:ListRoles",
"kms:DescribeKey",
"kms:ListAliases",
"lambda:ListFunctions",
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:DescribeLogGroups",
"logs:DescribeLogStreams",
"logs:GetLogEvents",
"logs:PutLogEvents",
"sns:ListTopics"
],
"Resource": "**",
}
,
{
"Effect": "Allow",
"Action": [  
"logs:CreateLogDelivery",
"logs:DeleteLogDelivery",
"logs:DescribeResourcePolicies",
"logs:GetLogDelivery",
"logs:ListLogDeliveries",
"logs:PutResourcePolicy",
"logs:UpdateLogDelivery"
],
"Resource": "**",
}
,
{
"Effect": "Allow",
"Action": [  
"ecr:SetRepositoryPolicy",
"ecr:CompleteLayerUpload",
"ecr:BatchDeleteImage",
"ecr:UploadLayerPart",
"ecr:DeleteRepositoryPolicy",
"ecr:InitiateLayerUpload",
"ecr:DeleteRepository",
"ecr:PutImage"
],
"Resource": "arn:aws:ecr::*:repository/*sagemaker*"
},
,
{
"Effect": "Allow",
"Action": [  
"codecommit:GitPull",
"codecommit:GitPush"
],
"Resource": [  
"arn:aws:codecommit::*:**sagemaker*",
"arn:aws:codecommit::*:*:sagemaker*"
]
{
"arn:aws:codecommit:*:*:*SageMaker*",
"arn:aws:codecommit:*:*:*Sagemaker*
},
{
"Effect": "Allow",
"Action": [
"secretsmanager:ListSecrets"
],
"Resource": "**"
},
{
"Effect": "Allow",
"Action": [
"secretsmanager:DescribeSecret",
"secretsmanager:GetSecretValue",
"secretsmanager:CreateSecret"
],
"Resource": [
"arn:aws:secretsmanager:*:*:secret:AmazonSageMaker-*"
]
},
{
"Effect": "Allow",
"Action": [
"secretsmanager:DescribeSecret",
"secretsmanager:GetSecretValue"
],
"Resource": "**",
"Condition": {
"StringEquals": {
"secretsmanager:ResourceTag/SageMaker": "true"
}
}
},
{
"Effect": "Allow",
"Action": [
"robomaker:CreateSimulationApplication",
"robomaker:DescribeSimulationApplication",
"robomaker:DeleteSimulationApplication"
],
"Resource": ["**"
]
},
{
"Effect": "Allow",
"Action": [
"robomaker:CreateSimulationJob",
"robomaker:DescribeSimulationJob",
"robomaker:CancelSimulationJob"
],
"Resource": ["**"
]
},
{
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject",
"s3:AbortMultipartUpload",
"s3:GetBucketCors",
"s3:PutBucketCors"
]


},
"Resource": [  
"arn:aws:s3:::*SageMaker*",  
"arn:aws:s3:::*Sagemaker*",  
"arn:aws:s3:::*sagemaker*",  
"arn:aws:s3:::*aws-glue*"
]
},
{  
"Effect": "Allow",  
"Action": [  
"s3:CreateBucket",  
"s3:GetBucketLocation",  
"s3:ListBucket",  
"s3:ListAllMyBuckets"
],  
"Resource": "*"
},

},
{  
"Effect": "Allow",  
"Action": [  
"s3:GetObject"
],  
"Resource": "*",  
"Condition": {  
"StringEqualsIgnoreCase": {  
"s3:ExistingObjectTag/SageMaker": "true"
  }
}
}
},
{  
"Effect": "Allow",  
"Action": [  
"lambda:InvokeFunction"
],  
"Resource": [  
"arn:aws:lambda::*:*:function:*SageMaker*",  
"arn:aws:lambda::*:*:function:*sagemaker*",  
"arn:aws:lambda::*:*:function:*Sagemaker*",  
"arn:aws:lambda::*:*:function:*LabelingFunction*"
]
}
},
{  
"Action": "iam:CreateServiceLinkedRole",  
"Effect": "Allow",  
"Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",  
"Condition": {  
"StringLike": {  
"iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
  }
}
}
},
{  
"Effect": "Allow",  
"Action": "iam:CreateServiceLinkedRole",  
"Resource": "*",  
"Condition": {  
"StringEquals": {  
"iam:AWSServiceName": "robomaker.amazonaws.com"
  }
}
}
],
{  
"Effect": "Allow",  
"Action": [  
"iam:GetAWSServiceRolePolicy",  
"iam:PassRole""
]
Identity-Based Policy Examples

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": [
                "arn:aws:iam::*:user/${aws:username}"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "sagemaker.amazonaws.com",
                        "glue.amazonaws.com",
                        "robomaker.amazonaws.com",
                        "states.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": [
                "arn:aws:iam::*:group/${aws:username}"
            ]
        }
    ]
}
```
Control Creation of Amazon SageMaker Resources with Condition Keys

Control fine-grained access to allow the creation of Amazon SageMaker resources by using Amazon SageMaker-specific condition keys. For information about using condition keys in IAM policies, see IAM JSON Policy Elements: Condition in the IAM User Guide.

For a list of Amazon SageMaker condition keys, see Condition Keys for Amazon SageMaker in the IAM User Guide.

The following examples show how to use Amazon SageMaker condition keys to control access.

Topics

• Control Access to Amazon SageMaker Resources by Using File System Condition Keys (p. 745)
• Restrict Training to a Specific VPC (p. 747)
• Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows (p. 747)
• Enforcing Encryption of Input Data (p. 749)
• Enforcing Encryption of Notebook Instance Storage Volume (p. 749)
• Enforcing Network Isolation for Training Jobs (p. 750)
• Enforcing a Specific Instance Type for Training Jobs (p. 750)
• Enforcing a Specific EI Accelerator for Training Jobs (p. 750)
• Enforce Disabling Internet Access and Root Access for Creating Notebook Instances (p. 751)

Control Access to Amazon SageMaker Resources by Using File System Condition Keys

Amazon SageMaker training provides a secure infrastructure for the training algorithm to run in, but for some cases you may want increased defense in depth. For example, you minimize the risk of running untrusted code in your algorithm, or you have specific security mandates in your organization. For these scenarios, you can use the following service-specific condition keys in the Condition element of an IAM policy to scope down the user to specific file systems, directories, access modes (read-write, read-only) and security groups:

Topics

• Restrict an IAM User to Specific Directories and Access Modes (p. 745)
• Restrict an IAM User to Specific File System (p. 746)

Restrict an IAM User to Specific Directories and Access Modes

The policy below restricts an IAM user to the /sagemaker/xgboost-dm/train and /sagemaker/xgboost-dm/validation directories of an EFS file system to ro (read-only) AccessMode:

Note

When a directory is allowed, all of its subdirectories are also accessible by the training algorithm. POSIX permissions are ignored.
Restrict an IAM User to Specific File System

To prevent a malicious algorithm using a user space client from accessing any file system directly in your account, you can restrict networking traffic by allowing ingress from a specific security group. In the following example, the IAM user can only use the specified security group to access the file system:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToLustreFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/fsx/sagemaker/xgboost/train"
                }
            }
        },
        {
            "Sid": "AccessToLustreFileSystemValidation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/fsx/sagemaker/xgboost/validation"
                }
            }
        }
    ]
}
```
Although the above example can restrict an algorithm to a specific file system, it does not prevent an algorithm from accessing any directory within that file system using the user space client. To mitigate this, you can:

- Ensure that the file system only contains data that you trust your IAM users to access
- Create an IAM role that restricts your IAM users to launching training jobs with algorithms from approved ECR repositories

For more information on how to use roles with Amazon SageMaker, see Amazon SageMaker Roles.

**Restrict Training to a Specific VPC**

Restrict an AWS user to creating training jobs from within a Amazon VPC. When a training job is created within a VPC, you can use VPC flow logs to monitor all traffic to and from the training cluster. For information about using VPC flow logs, see VPC Flow Logs in the Amazon Virtual Private Cloud User Guide.

The following policy enforces that a training job is created by an IAM user calling the section called “CreateTrainingJob” (p. 931) from within a VPC:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowFromVpc",
            "Effect": "Allow",
            "Action": ["sagemaker:CreateTrainingJob",
                        "sagemaker:CreateHyperParameterTuningJob"],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sagemaker:VpcSubnets": ["subnet-a1234"],
                    "sagemaker:VpcSecurityGroupIds": ["sg12345", "sg-67890"]
                },
                "Null": {
                    "sagemaker:VpcSubnets": "false",
                    "sagemaker:VpcSecurityGroupIds": "false"
                }
            }
        }
    ]
}
```

**Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows**

Amazon SageMaker Ground Truth and Amazon Augmented AI work teams fall into one of three workforce types: public (with Amazon Mechanical Turk), private, and vendor. To restrict
IAM user access to a specific work team using one of these types or the work team ARN, use the `sagemaker:WorkteamType` and/or the `sagemaker:WorkteamArn` condition keys. For the `sagemaker:WorkteamType` condition key, use string condition operators. For the `sagemaker:WorkteamArn` condition key, use Amazon Resource Name (ARN) condition operators. If the user attempts to create a labeling job with a restricted work team, Amazon SageMaker returns an access denied error.

The policies below demonstrate different ways to use the `sagemaker:WorkteamType` and `sagemaker:WorkteamArn` condition keys with appropriate condition operators and valid condition values.

The following example uses the `sagemaker:WorkteamType` condition key with the `StringEquals` condition operator to restrict access to a public work team. It accepts condition values in the following format: `workforcetype-crowd`, where `workforcetype` can equal `public`, `private`, or `vendor`.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "RestrictWorkteamType",
         "Effect": "Deny",
         "Action": "sagemaker:CreateLabelingJob",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sagemaker:WorkteamType": "public-crowd"
            }
         }
      }
   ]
}
```

The following policies show how to restrict access to a public work team using the `sagemaker:WorkteamArn` condition key. The first shows how to use it with a valid IAM regex-variant of the work team ARN and the `ArnLike` condition operator. The second shows how to use it with the `ArnEquals` condition operator and the work team ARN.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "RestrictWorkteamType",
         "Effect": "Deny",
         "Action": "sagemaker:CreateLabelingJob",
         "Resource": "*",
         "Condition": {
            "ArnLike": {
               "sagemaker:WorkteamArn": "arn:aws:sagemaker:*:*:workteam/public-crowd/*"
            }
         }
      }
   ]
}
```

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "RestrictWorkteamType",
         "Effect": "Deny",
         "Action": "sagemaker:CreateLabelingJob",
         "Resource": "*",
         "Condition": {
            "ArnEquals": {
               "sagemaker:WorkteamArn": "arn:aws:sagemaker:*:*:workteam/public-crowd/*"
            }
         }
      }
   ]
}
```
"Action": "sagemaker:CreateLabelingJob",
"Resource": "*",
"Condition": {
   "ArnEquals": {
   }
}
]
}

Enforcing Encryption of Input Data

The following policy restricts an IAM user to specify a AWS KMS key to encrypt input data when creating training, hyperparameter tuning, and labeling jobs by using the `sagemaker:VolumeKmsKey` condition key:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "EnforceEncryption",
         "Effect": "Allow",
         "Action": [
            "sagemaker:CreateTrainingJob",
            "sagemaker:CreateHyperParameterTuningJob",
            "sagemaker:CreateLabelingJob",
            "sagemaker:CreateFlowDefinition"
         ],
         "Resource": "*",
         "Condition": {
            "ArnEquals": {
               "sagemaker:VolumeKmsKey": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
         }
      }
   ]
}
```

Enforcing Encryption of Notebook Instance Storage Volume

The following policy restricts an IAM user to specify a AWS KMS key to encrypt the attached storage volume when creating or updating a notebook instance by using the `sagemaker:VolumeKmsKey` condition key:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "EnforceEncryption",
         "Effect": "Allow",
         "Action": ["sagemaker:CreateNotebookInstance"],
         "Resource": "*",
         "Condition": {
            "ArnLike": {
               "sagemaker:VolumeKmsKey": "*key/volume-kms-key-12345"
            }
         }
      }
   ]
}
```
Enforcing Network Isolation for Training Jobs

The following policy restricts an IAM user to enable network isolation when creating training jobs by using the `sagemaker:NetworkIsolation` condition key:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceIsolation",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateHyperParameterTuningJob"
      ],
      "Resource": "*",
      "Condition": {
        "Bool": {
          "sagemaker:NetworkIsolation": "true"
        }
      }
    }
  ]
}
```

Enforcing a Specific Instance Type for Training Jobs

The following policy restricts an IAM user to use a specific instance type when creating training jobs by using the `sagemaker:InstanceTypes` condition key:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceInstanceType",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateHyperParameterTuningJob"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "sagemaker:InstanceTypes": ["ml.c5.*"]
        }
      }
    }
  ]
}
```

Enforcing a Specific EI Accelerator for Training Jobs

The following policy restricts an IAM user to use a specific elastic inference (EI) accelerator, if an accelerator is provided, when creating or updating notebook instances and when creating endpoint configurations by using the `sagemaker:AcceleratorTypes` condition key:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceAccelerator",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:CreateEndpointConfiguration",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:UpdateEndpointConfiguration"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "sagemaker:AcceleratorTypes": ["ml.ei.*"]
        }
      }
    }
  ]
}
```
Identity-Based Policy Examples

Enforce Accelerator Type

You can enforce the use of a specific accelerator type when creating or updating a notebook instance. The following policy allows access to `sagemaker:CreateNotebookInstance`, `sagemaker:UpdateNotebookInstance`, and `sagemaker:CreateEndpointConfig` actions, but only if the `sagemaker:AcceleratorTypes` condition is met.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceAcceleratorType",
      "Effect": "Allow",
      "Action": ["sagemaker:CreateNotebookInstance", "sagemaker:UpdateNotebookInstance", "sagemaker:CreateEndpointConfig"],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "sagemaker:AcceleratorTypes": ["ml.eia1.medium"],
        }
      }
    }
  ]
}
```

Enforce Disabling Internet Access and Root Access for Creating Notebook Instances

You can disable both internet access and root access to notebook instances to help make them more secure. For information about controlling root access to a notebook instance, see Control Root Access to a Notebook Instance (p. 205). For information about disabling internet access for a notebook instance, see Connect a Notebook Instance to Resources in a VPC (p. 779).

The following policy requires an IAM user to disable network access when creating instance, and disable root access when creating or updating a notebook instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LockDownCreateNotebookInstance",
      "Effect": "Allow",
      "Action": ["sagemaker:CreateNotebookInstance"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:DirectInternetAccess": "Disabled",
          "sagemaker:RootAccess": "Disabled"
        },
        "Null": {
          "sagemaker:VpcSubnets": "false",
          "sagemaker:VpcSecurityGroupIds": "false"
        }
      }
    },
    {
      "Sid": "LockDownUpdateNotebookInstance",
      "Effect": "Allow",
      "Action": ["sagemaker:UpdateNotebookInstance"],
      "Resource": "*",
      "Condition": {
      }
    }
  ]
}
```
Control Access to the Amazon SageMaker API by Using Identity-based Policies

To control access to Amazon SageMaker API calls and calls to Amazon SageMaker hosted endpoints, use identity-based IAM policies.

Topics

- Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC (p. 752)

Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC

If you set up an interface endpoint in your VPC, individuals outside the VPC can still connect to the Amazon SageMaker API and runtime over the internet unless you attach an IAM policy that restricts access to calls coming from within the VPC to all users and groups that have access to your Amazon SageMaker resources. For information about creating a VPC interface endpoint for the Amazon SageMaker API and runtime, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 781).

Important

If you apply an IAM policy similar to one of the following, users can’t access the specified Amazon SageMaker APIs through the console.

To restrict access to only connections made from within your VPC, create an AWS Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every AWS Identity and Access Management user, group, or role used to access the Amazon SageMaker API or runtime.

Note

This policy allows connections only to callers within a subnet where you created an interface endpoint.

```json
{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpc": "vpc-111bbaaa"
                }
            }
        }
    ]
}
```
If you want to restrict access to the API to only calls made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```
{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": ["sagemaker:CreatePresignedNotebookInstanceUrl"],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        }
    ]
}
```

### Limit Access to Amazon SageMaker API and Runtime Calls by IP Address

To allow access to Amazon SageMaker API calls and runtime invocations only from IP addresses in a list that you specify, attach an IAM policy that denies access to the API unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access the API or runtime. For information about creating IAM policies, see Creating IAM Policies in the AWS Identity and Access Management User Guide. To specify the list of IP addresses that you want to have access to the API call, use the `IpAddress` condition operator and the `aws:SourceIp` condition context key. For information about IAM condition operators, see IAM JSON Policy Elements: Condition Operators in the AWS Identity and Access Management User Guide. For information about IAM condition context keys, see AWS Global Condition Context Keys.

For example, the following policy allows access to the `CreateTrainingJob` (p. 931) only from IP addresses in the ranges `192.0.2.0-192.0.2.255` and `203.0.113.0-203.0.113.255`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sagemaker:CreateTrainingJob",
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```
Limit Access to a Notebook Instance by IP Address

To allow access to a notebook instance only from IP addresses in a list that you specify, attach an IAM policy that denies access to CreatePresignedNotebookInstanceUrl (p. 924) unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access the notebook instance. For information about creating IAM policies, see Creating IAM Policies in the AWS Identity and Access Management User Guide. To specify the list of IP addresses that you want to have access to the notebook instance, use the IpAddress condition operator and the aws:SourceIp condition context key. For information about IAM condition operators, see IAM JSON Policy Elements: Condition Operators in the AWS Identity and Access Management User Guide. For information about IAM condition context keys, see AWS Global Condition Context Keys.

For example, the following policy allows access to a notebook instance only from IP addresses in the ranges 192.0.2.0-192.0.2.255 and 203.0.113.0-203.0.113.255:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24"
          ]
        }
      }
    }
  ]
}
```

The policy restricts access to both the call to CreatePresignedNotebookInstanceUrl and to the URL that the call returns. The policy also restricts access to opening a notebook instance in the console and is enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance.

**Note**

Using this method to filter by IP address is incompatible when connecting to Amazon SageMaker through a VPC interface endpoint. For information about restricting access to a notebook instance when connecting through a VPC interface endpoint, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 783).

Control Access to Amazon SageMaker Resources by Using Tags

Control access to groups of Amazon SageMaker resources by attaching tags to the resources and specifying ResourceTag conditions in IAM policies.

**Note**

Tag-based policies don't work to restrict the following API calls:

- ListAlgorithms
For example, suppose you've defined two different IAM groups, named DevTeam1 and DevTeam2, in your AWS account. Suppose also that you've created 10 notebook instances, 5 of which are used for one project, and 5 of which are used for a second project. You want to allow members of DevTeam1 to make API calls on notebook instances used for the first project, and members of DevTeam2 to make API calls on notebook instances used for the second project.

To control access to API calls (example)

1. Add a tag with the key Project and value A to the notebook instances used for the first project. For information about adding tags to Amazon SageMaker resources, see AddTags (p. 850).
2. Add a tag with the key Project and value B to the notebook instances used for the second project.
3. Create an IAM policy with a ResourceTag condition that denies access to the notebook instances used for the second project, and attach that policy to DevTeam1. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of Project and a value of B:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sagemaker:*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "Resource": "*",
            "Condition": {
```
"StringEquals": {
  "sagemaker:ResourceTag/Project": "B"
}
}
{
  "Effect": "Deny",
  "Action": [
    "sagemaker:CreateTags",
    "sagemaker:DeleteTags"
  ],
  "Resource": "*"
}
}

For information about creating IAM policies and attaching them to identities, see Controlling Access Using Policies in the AWS Identity and Access Management User Guide.

4. Create an IAM policy with a ResourceTag condition that denies access to the notebook instances used for the first project, and attach that policy to DevTeam2. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of Project and a value of A:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/Project": "A"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateTags",
        "sagemaker:DeleteTags"
      ],
      "Resource": "*"
    }
  ]
}
```

Require the Presence or Absence of Tags for API Calls

Require the presence or absence of specific tags or specific tag values by using RequestTag condition keys in an IAM policy. For example, if you want to require that every endpoint created by any member of an IAM group to be created with a tag with the key environment and value dev, create a policy as follows:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "sagemaker:ResourceTag/Project": "*"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateTags",
        "sagemaker:DeleteTags"
      ],
      "Resource": "*"
    }
  ]
}
```
Use Tags with Hyperparameter Tuning Jobs

You can add tags to a hyperparameter tuning job when you create the tuning job by specifying the tags as the Tags parameter when you call `CreateHyperParameterTuningJob` (p. 890). If you do this, the tags you specify for the hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches.

If you add tags to a hyperparameter tuning job by calling `AddTags` (p. 850), the tags you add are also added to any training jobs that the hyperparameter tuning job launches after you call `AddTags`, but are not added to training jobs the hyperparameter tuning jobs launched before you called `AddTags`. Similarly, when you remove tags from a hyperparameter tuning job by calling `DeleteTags` (p. 978), those tags are not removed from training jobs that the hyperparameter tuning job launched previously. Because of this, the tags associated with training jobs can be out of sync with the tags associated with the hyperparameter tuning job that launched them. If you use tags to control access to a hyperparameter tuning job and the training jobs it launches, you might want to keep the tags in sync. To make sure the tags associated with training jobs stay in sync with the tags associated with the hyperparameter tuning job that launched them, first call `ListTrainingJobsForHyperParameterTuningJob` (p. 1175) for the hyperparameter tuning job to get a list of the training jobs that the hyperparameter tuning job launched. Then, call `AddTags` or `DeleteTags` for the hyperparameter tuning job and for each of the training jobs in the list of training jobs to add or delete the same set of tags for all of the jobs. The following Python example demonstrates this:

```python
tuning_job_arn =
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName='MyTuningJob')['HyperParameterTuningJobArn']
smclient.add_tags(ResourceArn=tuning_job_arn, Tags=[{'Key': 'Env', 'Value': 'Dev'}])
training_jobs = smclient.list_training_jobs_for_hyper_parameter_tuning_job(
    HyperParameterTuningJobName='MyTuningJob')['TrainingJobSummaries']

for training_job in training_jobs:
    time.sleep(1) # Wait for 1 second between calls to avoid being throttled
    smclient.add_tags(ResourceArn=training_job['TrainingJobArn'], Tags=[{'Key': 'Env', 'Value': 'Dev'}])
```
Amazon SageMaker Roles

As a managed service, Amazon SageMaker performs operations on your behalf on the AWS hardware that is managed by Amazon SageMaker. Amazon SageMaker can perform only operations that the user permits.

An Amazon SageMaker user can grant these permissions with an IAM role (referred to as an execution role). The user passes the role when making these API calls: CreateNotebookInstance (p. 913), CreateHyperParameterTuningJob (p. 890), CreateProcessingJob (p. 926), CreateTrainingJob (p. 931), and CreateModel (p. 902).

You attach the following trust policy to the IAM role which grants Amazon SageMaker principal permissions to assume the role, and is the same for all of the execution roles:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The permissions that you need to grant to the role vary depending on the API that you call. The following sections explain these permissions.

**Note**

Instead of managing permissions by crafting a permission policy, you can use the AWS-managed AmazonSageMakerFullAccess permission policy. The permissions in this policy are fairly broad, to allow for any actions you might want to perform in Amazon SageMaker. For a listing of the policy including information about the reasons for adding many of the permissions, see AmazonSageMakerFullAccess Policy (p. 770). If you prefer to create custom policies and manage permissions to scope the permissions only to the actions you need to perform with the execution role, see the following topics.

For more information about IAM roles, see IAM Roles in the IAM User Guide.

**Topics**

- CreateNotebookInstance API: Execution Role Permissions (p. 758)
- CreateHyperParameterTuningJob API: Execution Role Permissions (p. 761)
- CreateProcessingJob API: Execution Role Permissions (p. 764)
- CreateTrainingJob API: Execution Role Permissions (p. 766)
- CreateModel API: Execution Role Permissions (p. 768)
- AmazonSageMakerFullAccess Policy (p. 770)

**CreateNotebookInstance API: Execution Role Permissions**

The permissions that you grant to the execution role for calling the CreateNotebookInstance API depend on what you plan to do with the notebook instance. If you plan to use it to invoke Amazon SageMaker APIs and pass the same role when calling the CreateTrainingJob and CreateModel APIs, attach the following permissions policy to the role:

```json
{
  
}
```
"Version": "2012-10-17",
"Statement": [
  {
"Effect": "Allow",
"Action": [
"sagemaker:*",
"ecr:GetAuthorizationToken",
"ecr:GetDownloadUrlForLayer",
"ecr:BatchGetImage",
"ecr:BatchCheckLayerAvailability",
"ecr:SetRepositoryPolicy",
"ecr:CompleteLayerUpload",
"ecr:BatchDeleteImage",
"ecr:UploadLayerPart",
"ecr:DeleteRepositoryPolicy",
"ecr:InitiateLayerUpload",
"ecr:DeleteRepository",
"ecr:PutImage",
"ecr:CreateRepository",
"cloudwatch:PutMetricData",
"cloudwatch:GetMetricData",
"cloudwatch:GetMetricStatistics",
"cloudwatch:ListMetrics",
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:DescribeLogStreams",
"logs:GetLogEvents",
"logs:GetLogEvents",
"s3:CreateBucket",
"s3:ListBucket",
"s3:GetBucketLocation",
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject",
"robomaker:CreateSimulationApplication",
"robomaker:DescribeSimulationApplication",
"robomaker:DeleteSimulationApplication",
"robomaker:CreateSimulationJob",
"robomaker:DescribeSimulationJob",
"robomaker:CancelSimulationJob",
"ec2:CreateVpcEndpoint",
"ec2:DescribeRouteTables",
"fsx:DescribeFileSystem",
"elasticfilesystem:DescribeMountTargets"
],
"Resource": "*",
},
{
"Effect": "Allow",
"Action": [
"codecommit:GitPull",
"codecommit:GitPush"
],
"Resource": [
"arn:aws:codecommit:*:*:*sagemaker*",
"arn:aws:codecommit:*:*:SageMaker*",
"arn:aws:codecommit:*:*:Sagemaker*"
]
},
{
"Effect": "Allow",
"Action": [
"iam:PassRole",

"Resource": "*",
"Condition": {

To tighten the permissions, limit them to specific Amazon S3 and Amazon ECR resources, by replacing "Resource": ", as follows:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "cloudwatch:PutMetricData",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object1",
                "arn:aws:s3:::outputbucket/path",
                "arn:aws:s3:::inputbucket/object2",
                "arn:aws:s3:::inputbucket/object3"
            ]
        }
    ]
}
```
If you plan to access other resources, such as Amazon DynamoDB or Amazon Relational Database Service, add the relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to the specific bucket that you specify as `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope `s3:GetObject`, `s3:PutObject`, and `s3:DeleteObject` permissions as follows:
  - Scope to the following values that you specify in a `CreateTrainingJob` request:
    - `InputDataConfig.DataSource.S3DataSource.S3Uri`
    - `OutputDataConfig.S3OutputPath`
  - Scope to the following values that you specify in a `CreateModel` request:
    - `PrimaryContainer.ModelDataUrl`
    - `SuplementalContainers.ModelDataUrl`
- Scope `ecr` permissions as follows:
  - Scope to the `AlgorithmSpecification.TrainingImage` value that you specify in a `CreateTrainingJob` request.
  - Scope to the `PrimaryContainer.Image` value that you specify in a `CreateModel` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

CreateHyperParameterTuningJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateHyperParameterTuningJob` API request, you can attach the following permission policy to the role:

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"cloudwatch:PutMetricData",
"logs:CreateLogStream",
"logs:PutLogEvents",
"logs:CreateLogGroup",
"ecr:BatchCheckLayerAvailability",
"ecr:GetDownloadUrlForLayer",
"ecr:BatchGetImage"
]
"Resource": [
"arn:aws:ecr:::repository/my-repo1",
"arn:aws:ecr:::repository/my-repo2",
"arn:aws:ecr:::repository/my-repo3"
]
}
}
```
Instead of specifying "Resource": "*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:PutMetricData",
            "logs:DescribeLogStreams",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:CreateLogGroup",
            "logs:DescribeLogStreams",
            "ecr:GetAuthorizationToken"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::inputbucket"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "s3:GetObject",
            "s3:PutObject"
         ],
         "Resource": [
            "arn:aws:s3:::inputbucket/object",
            "arn:aws:s3:::outputbucket/path"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "ecr:BatchCheckLayerAvailability",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage"
         ],
         "Resource": "arn:aws:ecr:::repository/my-repo"
      }
   ]
}
```
If the training container associated with the hyperparameter tuning job needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateHyperParameterTuningJob` request:
  - `InputDataConfig.DataSource.S3DataSource.S3Uri`
  - `OutputDataConfig.S3OutputPath`
- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateHyperParameterTuningJob` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your hyperparameter tuning job, add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [ "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
  ]
}
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [ "kms:Decrypt"
  ]
}
```

If you specify a KMS key in the output configuration of your hyperparameter tuning job, add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [ "kms:Encrypt"
  ]
}
```

If you specify a volume KMS key in the resource configuration of your hyperparameter tuning job, add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [ "kms:Encrypt", "kms:Decrypt"
  ]
}
```
CreateProcessingJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateProcessingJob` API request, you can attach the following permission policy to the role:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*
        },
    ]
}
```

Instead of specifying "Resource": "*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken",
                "s3:ListBucket",
            ],
            "Resource": ["arn:aws:s3:::inputbucket"
        },
    ]
}
```
If `CreateProcessingJob.AppSpecification.ImageUri` needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the `ProcessingInputs` in a `CreateProcessingJob` request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the objects that will be downloaded or uploaded in the `ProcessingInputs` and `ProcessingOutputConfig` in a `CreateProcessingJob` request.
- Scope Amazon ECR permissions to the registry path (`AppSpecification.ImageUri`) that you specify in a `CreateProcessingJob` request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your processing job, add the following permissions:

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:
If you specify a KMS key in the output configuration of your processing job, add the following permissions:

```json
{
  "Effect": "Allow",
  "Action": [
    "kms:Decrypt"
  ]
}
```

If you specify a volume KMS key in the resource configuration of your processing job, add the following permissions:

```json
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt"
  ]
}
```

**CreateTrainingJob API: Execution Role Permissions**

For an execution role that you can pass in a `CreateTrainingJob` API request, you can attach the following permission policy to the role:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

Instead of specifying "Resource": "*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "specific resource path"
    }
  ]
}
```
If `CreateTrainingJob.AlgorithmSpecifications.TrainingImage` needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateTrainingJob` request:
  - `InputDataConfig.DataSource.S3DataSource.S3Uri`
  - `OutputDataConfig.S3OutputPath`
- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateTrainingJob` request.

The `cloudwatch` and `logs` actions are applicable for 
"*" resources. For more information, see 
CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.
If you specify a private VPC for your training job, add the following permissions:

```
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateNetworkInterface",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface",
    "ec2:DeleteNetworkInterfacePermission",
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeVpcs",
    "ec2:DescribeDhcpOptions",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups"
  ]
}
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
  "Effect": "Allow",
  "Action": ["kms:Decrypt"
 }
```

If you specify a KMS key in the output configuration of your training job, add the following permissions:

```
{
  "Effect": "Allow",
  "Action": ["kms:Encrypt"
 }
```

If you specify a volume KMS key in the resource configuration of your training job, add the following permissions:

```
{
  "Effect": "Allow",
  "Action": ["kms:CreateGrant"
 }
```

**CreateModel API: Execution Role Permissions**

For an execution role that you can pass in a CreateModel API request, you can attach the following permission policy to the role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["cloudwatch:PutMetricData",
       "logs:CreateLogStream",
       "logs:PutLogEvents",
    ]
  
```
Instead of specifying "Resource": "*", you can scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket/object",
        "arn:aws:s3:::inputbucket/object"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": [
        "arn:aws:ecr:::repository/my-repo",
        "arn:aws:ecr:::repository/my-repo"
      ]
    }
  ]
}
```

If `CreateModel PrimaryContainer.Image` need to access other data sources, such as Amazon DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope S3 permissions to objects that you specify in the `PrimaryContainer.ModelDataUrl` in a `CreateModel (p. 902)` request.
• Scope Amazon ECR permissions to a specific registry path that you specify as the 

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see 
CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your model, add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ]
}
```

**AmazonSageMakerFullAccess Policy**

The AmazonSageMakerFullAccess managed policy includes all of the necessary permissions to perform 
most actions in Amazon SageMaker. You can use attach this policy to any role that you pass to an 
Amazon SageMaker execution role. You can also create more narrowly-scoped policies if you want more 
granular control of the permissions that you grant to your execution role.

The following list explains why some of the categories of permissions in the 
AmazonSageMakerFullAccess policy are needed.

**application-autoscaling**

Needed for automatically scaling an Amazon SageMaker real-time inference endpoint.

**aws-marketplace**

Needed to view AWS AI Marketplace subscriptions.

**cloudwatch**

Needed to post CloudWatch metrics, interact with alarms, and upload CloudWatch Logs logs in your 
account.

**codecommit**

Needed for AWS CodeCommit integration with Amazon SageMaker notebook instances.

**cognito**

Needed for Amazon SageMaker Ground Truth to define your private workforce and work teams.

**ec2**

Needed to manage elastic network interfaces when you specify a Amazon VPC for your Amazon 
SageMaker jobs and notebook instances.

**ec2:DescribeVpcs**

All Amazon SageMaker services launch Amazon EC2 instances and require this permission set.

**ecr**

Needed to pull and store Docker artifacts for training and inference. This is required only if you use 
your own container in Amazon SageMaker.
elastic-inference
   Needed to integrate Amazon Elastic Inference with Amazon SageMaker.

glue
   Needed for inference pipeline pre-processing from within Amazon SageMaker notebook instances.

groundtruthlabeling
   Needed for Amazon SageMaker Ground Truth.

iam:ListRoles
   Needed to give the Amazon SageMaker console access to list available roles.

kms
   Needed to give the Amazon SageMaker console access to list the available AWS KMS keys.

logs
   Needed to allow Amazon SageMaker jobs and endpoints to publish log streams.

AWS Managed (Predefined) Policies for Amazon SageMaker

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate which permissions are needed. For more information, see AWS Managed Policies in the IAM User Guide.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon SageMaker:

- **AmazonSageMakerReadOnly** – Grants read-only access to Amazon SageMaker resources.
- **AmazonSageMakerFullAccess** – Grants full access to Amazon SageMaker resources and the supported operations. (This does not provide unrestricted S3 access, but supports buckets/objects with specific sagemaker tags.)

The following AWS managed policies can also be attached to users in your account:

- **AdministratorAccess** – Grants all actions for all AWS services and for all resources in the account.
- **DataScientist** – Grants a wide range of permissions to cover most of the use cases (primarily for analytics and business intelligence) encountered by data scientists.

You can review these permissions policies by signing in to the IAM console and searching for them.

You can also create your own custom IAM policies to allow permissions for Amazon SageMaker actions and resources as you need them. You can attach these custom policies to the IAM users or groups that require them.

Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference

When you are setting up access control and writing a permissions policy that you can attach to an IAM identity (an identity-based policy), use the following as a reference. The each Amazon SageMaker API
operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's Action field, and you specify the resource value in the policy's Resource field.

**Note**
Except for the ListTags API, resource-level restrictions are not available on List- calls. Any user calling a List- API will see all resources of that type in the account.

To express conditions in your Amazon SageMaker policies, you can use AWS-wide condition keys. For a complete list of AWS-wide keys, see Available Keys in the IAM User Guide.

**Amazon SageMaker API and Required Permissions for Actions**

**API Operation:** AddTags (p. 850)
- Required Permissions (API Action): sagemaker:AddTags
- Resources: *

**API Operation:** CreateEndpoint (p. 875)
- Required Permissions (API Action): sagemaker:CreateEndpoint
- Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

**API Operation:** CreateEndpointConfig (p. 878)
- Required Permissions (API Action): sagemaker:CreateEndpointConfig
- Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

**API Operation:** CreateModel (p. 902)
- Required Permissions (API Action): sagemaker:CreateModel, iam:PassRole
- Resources: arn:aws:sagemaker:region:account-id:model/modelName

**API Operation:** CreateLabelingJob (p. 897)
- Required Permissions (API Action): sagemaker:CreateLabelingJob, iam:PassRole
- Resources: arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName

**API Operation:** CreateNotebookInstance (p. 913)
- Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

**API Operation:** CreateTrainingJob (p. 931)
- Required Permissions (API Action): sagemaker:CreateTrainingJob, iam:PassRole
- Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

**API Operation:** CreateWorkteam (p. 954)
API Operation: DeleteEndpoint (p. 963)
Required Permissions (API Action): sagemaker:DeleteEndpoint

API Operation: DeleteEndpointConfig (p. 965)
Required Permissions (API Action): sagemaker:DeleteEndpointConfig
Resources: arn:aws:sagemaker:region:account-id:endpoint/config/endpointConfigName

API Operation: DeleteModel (p. 970)
Required Permissions (API Action): sagemaker:DeleteModel
Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: DeleteNotebookInstance (p. 975)
Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: DeleteTags (p. 978)
Required Permissions (API Action): sagemaker:DeleteTags
Resources: *

API Operation: DeleteWorkteam (p. 986)
Required Permissions (API Action): sagemaker:DeleteWorkteam

API Operation: DescribeEndpoint (p. 1012)
Required Permissions (API Action): sagemaker:DescribeEndpoint
Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: DescribeEndpointConfig (p. 1015)
Required Permissions (API Action): sagemaker:DescribeEndpointConfig
Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: DescribeLabelingJob (p. 1034)
Required Permissions (API Action): sagemaker:DescribeLabelingJob
Resources: arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName

API Operation: DescribeModel (p. 1040)
Required Permissions (API Action): sagemaker:DescribeModel
Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: DescribeNotebookInstance (p. 1051)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: DescribeSubscribedWorkteam (p. 1064)

Required Permissions (API Action): sagemaker:DescribeSubscribedWorkteam, aws-marketplace:ViewSubscriptions


API Operation: DescribeTrainingJob (p. 1066)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

API Operation: DescribeWorkteam (p. 1091)

Required Permissions (API Action): sagemaker:DescribeWorkteam


API Operation: CreatePresignedNotebookInstanceUrl (p. 924)

Required Permissions (API Action): sagemaker:CreatePresignedNotebookInstanceUrl

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: InvokeEndpoint (p. 1260)

Required Permissions (API Action): sagemaker:InvokeEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: ListEndpointConfigs (p. 1118)

Required Permissions (API Action): sagemaker:ListEndpointConfigs

Resources: *

API Operation: ListEndpoints (p. 1121)

Required Permissions (API Action): sagemaker:ListEndpoints

Resources: *

API Operation: ListLabelingJobs (p. 1137)

Required Permissions (API Action): sagemaker:ListLabelingJobs

Resources: *

API Operation: ListLabelingJobsForWorkteam (p. 1141)

Required Permissions (API Action): sagemaker:ListLabelingJobsForWorkteam

Resources: *

API Operation: ListModels (p. 1147)

Required Permissions (API Action): sagemaker:ListModels
Resources: *

**API Operation: ListNotebookInstances (p. 1161)**

Required Permissions (API Action): sagemaker:ListNotebookInstances

Resources: *

**API Operation: ListSubscribedWorkteams (p. 1168)**


**API Operation: ListTags (p. 1170)**

Required Permissions (API Action): sagemaker:ListTags

Resources: *

**API Operation: ListTrainingJobs (p. 1172)**

Required Permissions (API Action): sagemaker:ListTrainingJobs

Resources: *

**API Operation: ListWorkteams (p. 1191)**

Required Permissions (API Action): sagemaker:ListWorkteams


**API Operation: StartNotebookInstance (p. 1208)**


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

**API Operation: StopLabelingJob (p. 1216)**

Required Permissions (API Action): sagemaker:StopLabelingJob

Resources: arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName

**API Operation: StopNotebookInstance (p. 1220)**

Required Permissions (API Action): sagemaker:StopNotebookInstance

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

**API Operation: StopTrainingJob (p. 1224)**

Required Permissions (API Action): sagemaker:StopTrainingJob

Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

**API Operation: UpdateEndpoint (p. 1233)**

Required Permissions (API Action): sagemaker:UpdateEndpoints

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName
API Operation: UpdateNotebookInstance (p. 1242)

Required Permissions (API Action): sagemaker:UpdateNotebookInstance, iam:PassRole

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: UpdateWorkteam (p. 1257)

Required Permissions (API Action): sagemaker:UpdateWorkteam


Troubleshooting Amazon SageMaker Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon SageMaker and IAM.

Topics
- I Am Not Authorized to Perform an Action in Amazon SageMaker (p. 776)
- I Am Not Authorized to Perform iam:PassRole (p. 776)
- I Want to View My Access Keys (p. 777)
- I'm an Administrator and Want to Allow Others to Access Amazon SageMaker (p. 777)
- I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources (p. 777)

I Am Not Authorized to Perform an Action in Amazon SageMaker

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a training job but does not have sagemaker:DescribeTrainingJob permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: sagemaker:DescribeTrainingJob on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the TrainingJob resource using the sagemaker:DescribeTrainingJob action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon SageMaker.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.
The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in Amazon SageMaker. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtNfEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the IAM User Guide.

I'm an Administrator and Want to Allow Others to Access Amazon SageMaker

To allow others to access Amazon SageMaker, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon SageMaker.

To get started right away, see Creating Your First IAM Delegated User and Group in the IAM User Guide.

I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon SageMaker supports these features, see How Amazon SageMaker Works with IAM (p. 732).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the IAM User Guide.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the IAM User Guide.
Logging and Monitoring

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see Monitor Amazon SageMaker with Amazon CloudWatch (p. 712).

Amazon CloudWatch Logs enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 719).

AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Log Amazon SageMaker API Calls with AWS CloudTrail (p. 720).

Note
CloudTrail does not monitor calls to InvokeEndpoint (p. 1260).

You can create rules in Amazon CloudWatch Events to react to status changes in status in an Amazon SageMaker training, hyperparameter tuning, or batch transform job. For more information, see React to Amazon SageMaker Job Status Changes with CloudWatch Events (p. 723).

Compliance Validation for Amazon SageMaker

Third-party auditors assess the security and compliance of Amazon SageMaker as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon SageMaker is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
Resilience in Amazon SageMaker

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon SageMaker offers several features to help support your data resiliency and backup needs.

Infrastructure Security in Amazon SageMaker

As a managed service, Amazon SageMaker is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon SageMaker through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Topics
- Connect a Notebook Instance to Resources in a VPC (p. 779)
- Training and Inference Containers Run in Internet-Free Mode (p. 780)
- Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities (p. 781)
- Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 781)
- Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC (p. 786)
- Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC (p. 792)
- Give Batch Transform Jobs Access to Resources in Your Amazon VPC (p. 795)

Connect a Notebook Instance to Resources in a VPC

Amazon SageMaker notebook instances are internet-enabled by default. This allows you to download popular packages and notebooks, customize your development environment, and work efficiently. However, if you connect a notebook instance to your VPC, the notebook instance provides an additional avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally
install on the computer (in the form of a publicly available notebook or a publicly available source code library) could access your data. If you do not want Amazon SageMaker to provide internet access to your notebook instance, you can disable direct internet access when you specify a VPC for your notebook instance. If you disable direct internet access, the notebook instance won't be able to train or host models unless your VPC has an interface endpoint (PrivateLink) or a NAT gateway and your security groups allow outbound connections. For information about creating a VPC interface endpoint to use PrivateLink for your notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 783). For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide. For information about security groups, see Security Groups for Your VPC.

Notebook Instances Provide the Best Experience for a Single User

An Amazon SageMaker notebook instance is designed to work best for an individual user. It is designed to give data scientists and other users the most power for managing their development environment. A notebook instance user has root access for installing packages and other pertinent software. We recommend that you exercise judgement when granting individuals access to notebook instances that are attached to a VPC that contains sensitive information. For example, you might grant a user access to a notebook instance with an IAM policy, as in the following example:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
    }
  ]
}
```

Training and Inference Containers Run in Internet-Free Mode

Amazon SageMaker training and deployed inference containers are internet-enabled by default. This allows containers to access external services and resources on the public internet as part of your training and inference workloads. However, this offers an avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the container (in the form of a publicly available source code library) could access your data and transfer it to a remote host. If you use an Amazon VPC by specifying a value for the VpcConfig parameter when you call CreateTrainingJob (p. 931), CreateHyperParameterTuningJob (p. 890), or CreateModel (p. 902), you can protect your data and resources by managing security groups and restricting internet access from your VPC. However, this comes at the cost of additional network configuration, and has the risk of configuring your network incorrectly. If you do not want Amazon SageMaker to provide external network access to your training or inference containers, you can enable network isolation when you create your training job or model by setting the value of the EnableNetworkIsolation parameter to True when you call CreateTrainingJob (p. 931), CreateHyperParameterTuningJob (p. 890), or CreateModel (p. 902). If you enable network isolation, the containers are not able to make any outbound network calls, even to other AWS services such as Amazon S3. Additionally, no AWS credentials are made available to the container runtime environment. In the case of a training job with multiple instances, network inbound and outbound traffic is limited to the peers of each training container. Amazon SageMaker still performs download and upload operations against Amazon S3 using your Amazon SageMaker Execution Role in isolation from the training or inference container. Network
isolation is required for training jobs and models run using resources from AWS Marketplace. Network isolation can be used in conjunction with a VPC. In this scenario, download and upload of customer data and model artifacts are routed via your VPC subnet. However, the training and inference containers themselves continue to be isolated from the network, and do not have access to any resource within your VPC or on the internet.

Network isolation is not supported by the following managed Amazon SageMaker containers as they require access to Amazon S3:

- Chainer
- PyTorch
- Scikit-learn
- Amazon SageMaker Reinforcement Learning

Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities

To meet our security requirements, algorithms and model packages listed in AWS Marketplace are scanned for Common Vulnerabilities and Exposures (CVE). CVE is a list of publicly known information about security vulnerability and exposure. The National Vulnerability Database (NVD) provides CVE details such as severity, impact rating, and fix information. Both CVE and NVD are available for public consumption and free for security tools and services to use. For more information, see http://cve.mitre.org/about/faqs.html#what_is_cve.

Connect to Amazon SageMaker Through a VPC Interface Endpoint

You can connect directly to the Amazon SageMaker API or to the Amazon SageMaker Runtime through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the Amazon SageMaker API or Runtime is conducted entirely and securely within the AWS network.

The Amazon SageMaker API and Runtime support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

The VPC interface endpoint connects your VPC directly to the Amazon SageMaker API or Runtime without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Amazon SageMaker API or Runtime.

You can create an interface endpoint to connect to Amazon SageMaker or to Amazon SageMaker Runtime with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see Creating an Interface Endpoint.

After you have created a VPC endpoint, you can use the following example CLI commands that use the endpoint-url parameter to specify interface endpoints to the Amazon SageMaker API or Runtime:

```bash
aws sagemaker list-notebook-instances --endpoint-url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker list-training-jobs --endpoint-url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```
Connect to Amazon SageMaker Through a VPC Interface Endpoint

aws sagemaker-runtime invoke-endpoint --endpoint-url VPC_Endpoint_ID.runtime.sagemaker.Region.vpce.amazonaws.com \
--endpoint-name Endpoint_Name \
--body "Endpoint_Body" \
--content-type "Content_Type" \
Output_File

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint URL. The Amazon SageMaker API DNS hostname that the CLI and Amazon SageMaker SDK use by default (https://api.sagemaker.Region.amazonaws.com) resolves to your VPC endpoint. Similarly, the Amazon SageMaker Runtime DNS hostname that the CLI and Amazon SageMaker Runtime SDK use by default (https://runtime.sagemaker.Region.amazonaws.com) resolves to your VPC endpoint.

The Amazon SageMaker API and Runtime support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available. Amazon SageMaker supports making calls to all of its Actions (p. 843) inside your VPC. The result AuthorizedUrl from the CreatePresignedNotebookInstanceUrl (p. 924) is not supported by Private Link. For information about how to enable PrivateLink for the authorized URL that users use to connect to a notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 783).

To learn more about AWS PrivateLink, see the AWS PrivateLink documentation. Refer to VPC Pricing for the price of VPC Endpoints. To learn more about VPC and Endpoints, see Amazon VPC. For information about how to use identity-based AWS Identity and Access Management policies to restrict access to the Amazon SageMaker API and runtime, see Control Access to the Amazon SageMaker API by Using Identity-based Policies (p. 752).

Create a VPC Endpoint Policy for Amazon SageMaker

You can create a policy for Amazon VPC endpoints for Amazon SageMaker to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the Amazon VPC User Guide.

**Note**

VPC endpoint policies aren't supported for Federal Information Processing Standard (FIPS) Amazon SageMaker runtime endpoints for InvokeEndpoint (p. 1260).

The following example VPC endpoint policy specifies that all users who have access to the VPC interface endpoint are allowed to invoke the Amazon SageMaker hosted endpoint named myEndpoint.

```json
{
"Statement": [
{
"Action": "sagemaker:InvokeEndpoint",
"Effect": "Allow",
"Principal": "*

}]
}
```

In this example, the following are denied:

- Other Amazon SageMaker API actions, such as sagemaker:CreateEndpoint and sagemaker:CreateTrainingJob.
• Invoking Amazon SageMaker hosted endpoints other than myEndpoint.

Note
In this example, users can still take other Amazon SageMaker API actions from outside the VPC. For information about how to restrict API calls to those from within the VPC, see Control Access to the Amazon SageMaker API by Using Identity-based Policies (p. 752).

Connect to a Notebook Instance Through a VPC Interface Endpoint

You can connect to your notebook instance from your VPC through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the notebook instance is conducted entirely and securely within the AWS network.

Amazon SageMaker notebook instances support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

Note
Before you create an interface VPC endpoint to connect to a notebook instance, create an interface VPC endpoint to connect to the Amazon SageMaker API. That way, when users call CreatePresignedNotebookInstanceUrl (p. 924) to get the URL to connect to the notebook instance, that call also goes through the interface VPC endpoint. For information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 781).

You can create an interface endpoint to connect to your notebook instance with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see Creating an Interface Endpoint. Make sure that you create an interface endpoint for all of the subnets in your VPC from which you want to connect to the notebook instance.

When you create the interface endpoint, specify aws.sagemaker.region.notebook as the service name. After you create a VPC endpoint, enable private DNS for your VPC endpoint. Anyone using the Amazon SageMaker API, the AWS CLI, or the console to connect to the notebook instance from within the VPC will connect to the notebook instance through the VPC endpoint instead of the public internet.

Amazon SageMaker notebook instances support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available.

Topics
• Connect Your Private Network to Your VPC (p. 783)
• Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances (p. 783)
• Restrict Access to Connections from Within Your VPC (p. 784)

Connect Your Private Network to Your VPC

To connect to your notebook instance through your VPC, you either have to connect from an instance that is inside the VPC, or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN Connections in the Amazon Virtual Private Cloud User Guide. For information about AWS Direct Connect, see Creating a Connection in the AWS Direct Connect User Guide.

Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances

You can create a policy for Amazon VPC endpoints for Amazon SageMaker notebook instances to specify the following:
• The principal that can perform actions.
• The actions that can be performed.
• The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the Amazon VPC User Guide.

The following example of a VPC endpoint policy specifies that all users that have access to the endpoint are allowed to access the notebook instance named myNotebookInstance.

```json
{
   "Statement": [
      {
         "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
         "Effect": "Allow",
         "Principal": "*
      }
   ]
}
```

Access to other notebook instances is denied.

**Restrict Access to Connections from Within Your VPC**

Even if you set up an interface endpoint in your VPC, individuals outside the VPC can connect to the notebook instance over the internet.

**Important**

If you apply an IAM policy similar to one of the following, users can't access the specified Amazon SageMaker APIs or the notebook instance through the console.

To restrict access to only connections made from within your VPC, create an AWS Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every AWS Identity and Access Management user, group, or role used to access the notebook instance.

**Note**

This policy allows connections only to callers within a subnet where you created an interface endpoint.

```json
{
   "Id": "notebook-example-1",
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Enable Notebook Access",
         "Effect": "Allow",
         "Action": ["sagemaker:CreatePresignedNotebookInstanceUrl", "sagemaker:DescribeNotebookInstance"],
         "Resource": "*
         "Condition": {
            "StringEquals": {
               "aws:SourceVpc": "vpc-11bbbaaa"
            }
         }
      }
   ]
}
```
If you want to restrict access to the notebook instance to only connections made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```json
{
   "Id": "notebook-example-1",
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Enable Notebook Access",
         "Effect": "Allow",
         "Action": [
            "sagemaker:CreatePresignedNotebookInstanceUrl",
            "sagemaker:DescribeNotebookInstance"
         ],
         "Resource": "*",
         "Condition": {
            "ForAnyValue:StringEquals": {
               "aws:sourceVpce": [
                  "vpce-111bbccc",
                  "vpce-111bbddd"
               ]
            }
         }
      }
   ]
}
```

Both of these policy examples assume that you have also created an interface endpoint for the Amazon SageMaker API. For more information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 781). In the second example, one of the values for `aws:SourceVpce` is the ID of the interface endpoint for the notebook instance. The other is the ID of the interface endpoint for the Amazon SageMaker API.

The policy examples here include DescribeNotebookInstance (p. 1051) because typically you would call DescribeNotebookInstance to make sure that the NotebookInstanceStatus is InService before you try to connect to it. For example:

```bash
aws sagemaker describe-notebook-instance --notebook-instance-name myNotebookInstance
```

```json
{
   "NotebookInstanceArn":
   "NotebookInstanceName": "myNotebookInstance",
   "NotebookInstanceStatus": "InService",
   "Url": "mynotebookinstance.notebook.us-west-2.sagemaker.aws",
   "InstanceType": "ml.m4.xlarge",
   "RoleArn":
   "arn:aws:iam::1234567890ab:role/service-role/AmazonSageMaker-ExecutionRole-12345678T123456",
   "LastModifiedTime": 1540334777.501,
   "CreationTime": 1523050674.078,
   "DirectInternetAccess": "Disabled"
}
```

```bash
aws sagemaker create-presigned-notebook-instance-url --notebook-instance-name myNotebookInstance
```

```json
{
```
Amazon SageMaker Developer Guide
Give Amazon SageMaker Processing Jobs Access to Resources in Your Amazon VPC

"AuthorizedUrl": "https://mynotebookinstance.notebook.us-west-2.sagemaker.aws?
authToken=AuthToken
"

For both of these calls, if you did not enable private DNS hostnames for your VPC endpoint, or if you
are using a version of the AWS SDK that was released before August 13, 2018, you must specify the
endpoint URL in the call. For example, the call to create-presigned-notebook-instance-url
would be:

```bash
aws sagemaker create-presigned-notebook-instance-url
--notebook-instance-name myNotebookInstance --endpoint-url
VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

Connect Your Private Network to Your VPC

To call the Amazon SageMaker API and runtime through your VPC, you have to connect from an instance
that is inside the VPC or connect your private network to your VPC by using an Amazon Virtual Private
Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN Connections in
the Amazon Virtual Private Cloud User Guide. For information about AWS Direct Connect, see Creating a
Connection in the AWS Direct Connect User Guide.

Give Amazon SageMaker Processing Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs processing jobs in an Amazon Virtual Private Cloud by default. However,
processing containers access AWS resources—such as the Amazon S3 buckets where you store data—
over the internet.

To control access to your data and processing containers, we recommend that you create a private VPC
and configure it so that they aren’t accessible over the internet. For information about creating and
configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC
helps to protect your processing containers and data because you can configure your VPC so that it is not
connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your
processing containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon
VPC User Guide.

You specify your private VPC configuration when you create processing jobs by specifying subnets and
security groups. When you specify the subnets and security groups, Amazon SageMaker creates elastic
network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow
your processing containers to connect to resources in your VPC. For information about ENIs, see Elastic
Network Interfaces in the Amazon VPC User Guide.

Configure a Processing Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the NetworkConfig.VpcConfig
request parameter of the CreateProcessingJob (p. 926) API, or provide this information when you
create a processing job in the Amazon SageMaker console. Amazon SageMaker uses this information to
create ENIs and attach them to your processing containers. The ENIs provide your processing containers
with a network connection within your VPC that is not connected to the internet. They also enable your
processing job to connect to resources in your private VPC.

The following is an example of the VpcConfig parameter that you include in your call to
CreateProcessingJob:

```json
VpcConfig: {
    "Subnets": [
Configure Your Private VPC for Amazon SageMaker Processing

When configuring the private VPC for your Amazon SageMaker processing jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.

Topics
- Ensure That Subnets Have Enough IP Addresses (p. 787)
- Create an Amazon S3 VPC Endpoint (p. 787)
- Use a Custom Endpoint Policy to Restrict Access to S3 (p. 787)
- Configure Route Tables (p. 788)
- Configure the VPC Security Group (p. 788)
- Connect to Resources Outside Your VPC (p. 788)

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a processing job. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that processing containers don’t have access to the internet, they can’t connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your processing containers to access the buckets where you store your data. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Endpoints, then choose Create Endpoint
3. For Service Name, choose com.amazonaws.region.s3, where region is the name of the region where your VPC resides.
4. For VPC, choose the VPC you want to use for this endpoint.
5. For Configure route tables, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For Policy, choose Full Access to allow full access to the S3 service by any user or service within the VPC. Choose Custom to restrict access further. For information, see Use a Custom Endpoint Policy to Restrict Access to S3 (p. 787).

Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for
Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Processing Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the processing container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
  "Statement": [
    {
      "Sid": "AmazonLinuxAMIRepositoryAccess",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::packages.*.amazonaws.com/**",
        "arn:aws:s3:::repo.*.amazonaws.com/**"
      ]
    }
  ]
}
```
```
{
  "Statement": [
    {
      "Sid": "AmazonLinux2AMIRepositoryAccess",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::amazonlinux.*.amazonaws.com/**"
      ]
    }
  ]
}
```

Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your processing jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

Configure the VPC Security Group

In distributed processing, you must allow communication between the different containers in the same processing job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see Security Group Rules.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, processing jobs that use that VPC do not have access to resources outside your VPC. If your processing job needs access to resources outside your VPC, provide access with one of the following options:
Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud by default. However, training containers access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To control access to your data and training containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

You specify your private VPC configuration when you create training jobs by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your training containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

Note
For training jobs, you can configure only subnets with a default tenancy VPC in which your instance runs on shared hardware. For more information on the tenancy attribute for VPCs, see Dedicated Instances.

Configure a Training Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateTrainingJob (p. 931) API, or provide this information when you create a training job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your training containers. The ENIs provide your training containers with a network connection within your VPC that is not connected to the internet. They also enable your training job to connect to resources in your private VPC.

The following is an example of the VpcConfig parameter that you include in your call to CreateTrainingJob:

```json
VpcConfig: {
    "Subnets": [
        "subnet-0123456789abcdef0",
        "subnet-0123456789abcdef1",
        "subnet-0123456789abcdef2"
    ],
    "SecurityGroupIds": [
        "sg-0123456789abcdef0"
    ]
}
```
Configure Your Private VPC for Amazon SageMaker Training

When configuring the private VPC for your Amazon SageMaker training jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.

Topics

• Ensure That Subnets Have Enough IP Addresses (p. 790)
• Create an Amazon S3 VPC Endpoint (p. 790)
• Use a Custom Endpoint Policy to Restrict Access to S3 (p. 790)
• Configure Route Tables (p. 791)
• Configure the VPC Security Group (p. 791)
• Connect to Resources Outside Your VPC (p. 791)

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a training job. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that training containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your training data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your training containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose Endpoints, then choose Create Endpoint.
3. For Service Name, choose com.amazonaws.region.s3, where region is the name of the region where your VPC resides.
4. For VPC, choose the VPC you want to use for this endpoint.
5. For Configure route tables, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For Policy, choose Full Access to allow full access to the S3 service by any user or service within the VPC. Choose Custom to restrict access further. For information, see Use a Custom Endpoint Policy to Restrict Access to S3 (p. 790).

Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Training Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository,
create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```json
{
  "Statement": [
    {
      "Sid": "AmazonLinuxAMIRepositoryAccess",
      "Principal": "*",
      "Action": [ "s3:GetObject"
      ],
      "Effect": "Deny",
      "Resource": [ "arn:aws:s3:::packages.*.amazonaws.com/*", "arn:aws:s3:::repo.*.amazonaws.com/*"
      ]
    }
  ]
}
{
  "Statement": [
    {
      "Sid": "AmazonLinux2AMIRepositoryAccess",
      "Principal": "*",
      "Action": [ "s3:GetObject"
      ],
      "Effect": "Deny",
      "Resource": [ "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
      ]
    }
  ]
}
```

**Configure Route Tables**

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don’t use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your training jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

**Configure the VPC Security Group**

In distributed training, you must allow communication between the different containers in the same training job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group For information, see Security Group Rules.

**Connect to Resources Outside Your VPC**

If you configure your VPC so that it doesn’t have internet access, training jobs that use that VPC do not have access to resources outside your VPC. If your training job needs access to resources outside your VPC, provide access with one of the following options:

- If your training job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the Amazon VPC User Guide. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

- If your training job needs access to an AWS service that doesn’t support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow
outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide.

Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC

Amazon SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To avoid making your data and model containers accessible over the internet, we recommend that you create a private VPC and configure it to control access to them. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

Configure a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateModel API, or provide this information when you create a model in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your model to connect to resources in your private VPC.

Note
You must create at least two subnets in different availability zones in your private VPC, even if you have only one hosting instance.

The following is an example of the VpcConfig parameter that you include in your call to CreateModel:

```json
VpcConfig: {
   "Subnets": [
      "subnet-0123456789abcdef0",
      "subnet-0123456789abcdef1",
      "subnet-0123456789abcdef2"
   ],
   "SecurityGroupIds": [
      "sg-0123456789abcdef0"
   ]
}
```

Configure Your Private VPC for Amazon SageMaker Hosting

When configuring the private VPC for your Amazon SageMaker models, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.

Topics
Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each model instance. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

To create an Amazon S3 VPC endpoint:

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose `com.amazonaws.region.s3`, where `region` is the name of the AWS Region where your VPC resides.
4. For **VPC**, choose the VPC that you want to use for this endpoint.
5. For **Configure route tables**, choose the route tables that the endpoint will use. The VPC service automatically adds a route to each route table that you choose that points Amazon S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the Amazon S3 service by any user or service within the VPC. To restrict access further, choose **Custom**. For more information, see Use a Custom Endpoint Policy to Restrict Access to Amazon S3 (p. 793).

Use a Custom Endpoint Policy to Restrict Access to Amazon S3

The default endpoint policy allows full access to Amazon Simple Storage Service (Amazon S3) for any user or service in your VPC. To further restrict access to Amazon S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3.

You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Model Container with a Custom Endpoint Policy

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from those repositories, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```json
{
    "Statement": [
```
Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies

The SageMakerFullAccess managed policy includes the permissions that you need to use models configured for Amazon VPC access with an endpoint. These permissions allow Amazon SageMaker to create an elastic network interface and attach it to model containers running in a VPC. If you use your own IAM policy, you must add the following permissions to that policy to use models configured for VPC access.

```json
```

For more information about the SageMakerFullAccess managed policy, see AmazonSageMakerFullAccess Policy (p. 770).
Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don’t use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your models resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn’t have internet access, models that use that VPC do not have access to resources outside your VPC. If your model needs access to resources outside your VPC, provide access with one of the following options:

- If your model needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the Amazon VPC User Guide. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.
- If your model needs access to an AWS service that doesn’t support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide.

Give Batch Transform Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs batch transform jobs in an Amazon Virtual Private Cloud by default. However, model containers access AWS resources—such as the Amazon S3 buckets where you store your data and model artifacts—over the internet.

To control access to your model containers and data, we recommend that you create a private VPC and configure it so that they aren’t accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC helps to protect your model containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your model containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. You then specify the same model when you create a transform job. When you specify the subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

Configure a Batch Transform Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateModel (p. 902) API, or provide this information when you create a transform job in the Amazon SageMaker console. Then specify the same model in the ModelName request parameter of the CreateTransformJob (p. 939) API, or when you create a transform job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your transform job to connect to resources in your private VPC.
The following is an example of the VpcConfig parameter that you include in your call to CreateModel:

```json
VpcConfig: {
    "Subnets": [
        "subnet-0123456789abcdef0",
        "subnet-0123456789abcdef1",
        "subnet-0123456789abcdef2"
    ],
    "SecurityGroupIds": [
        "sg-0123456789abcdef0"
    ]
}
```

**Configure Your Private VPC for Amazon SageMaker Batch Transform**

When configuring the private VPC for your Amazon SageMaker batch transform jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.

**Topics**

- Ensure That Subnets Have Enough IP Addresses (p. 796)
- Create an Amazon S3 VPC Endpoint (p. 796)
- Use a Custom Endpoint Policy to Restrict Access to S3 (p. 797)
- Configure Route Tables (p. 797)
- Configure the VPC Security Group (p. 797)
- Connect to Resources Outside Your VPC (p. 798)

**Ensure That Subnets Have Enough IP Addresses**

Your VPC subnets should have at least two private IP addresses for each instance in a transform job. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

**Create an Amazon S3 VPC Endpoint**

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

**To create an S3 VPC endpoint:**

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose `com.amazonaws.region.s3`, where `region` is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see Use a Custom Endpoint Policy to Restrict Access to S3 (p. 797).
Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see Using Endpoint Policies for Amazon S3. You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see Using Amazon S3 Bucket Policies.

Restrict Package Installation on the Model Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```json
{
   "Statement": [
      {
         "Sid": "AmazonLinuxAMIRepositoryAccess",
         "Principal": "*",
         "Action": ["s3:GetObject"],
         "Effect": "Deny",
         "Resource": [
            "arn:aws:s3:::packages.*.amazonaws.com/*",
            "arn:aws:s3:::repo.*.amazonaws.com/*"
         ]
      }
   ],
   "Statement": [
      {
         "Sid": "AmazonLinux2AMIRepositoryAccess",
         "Principal": "*",
         "Action": ["s3:GetObject"],
         "Effect": "Deny",
         "Resource": ["arn:aws:s3:::amazonlinux.*.amazonaws.com/*"]
      }
   ]
}
```

Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your batch transform jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

Configure the VPC Security Group

In distributed batch transform, you must allow communication between the different containers in the same batch transform job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see Security Group Rules.
Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, batch transform jobs that use that VPC do not have access to resources outside your VPC. If your batch transform job needs access to resources outside your VPC, provide access with one of the following options:

- If your batch transform job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the Amazon VPC User Guide. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.
- If your batch transform job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide.
Limits and Supported Regions

For Amazon SageMaker service limits, see Amazon SageMaker Limits.
For information about requesting limit increases for AWS resources, see AWS Service Limits.
For a list of the AWS Regions supporting Amazon SageMaker, see Amazon SageMaker Regions.

Topics
• Supported Instance Types and Availability Zones (p. 799)

Supported Instance Types and Availability Zones

The tables in this topic list the availability of instance types for each AWS Region and Availability Zone for the following Amazon SageMaker components.

• Notebook instances – listed as Notebook in the tables
• Training jobs – listed as Training in the tables
• Batch transform jobs – listed as Batch in the tables
• Hosted endpoints – listed as Endpoint in the tables

Not all components are supported on each instance type in each Availability Zone.

There is one table for each AWS Region. In each table, the Amazon SageMaker components that support each instance type are listed for each Availability Zone. If no components support the instance type in an Availability Zone, the cell contains "None". If all components support the instance type in an Availability Zone, the cell contains "All".

To create a component on a specific instance type, you must specify the Availability Zone ID, which is listed in the header row of each table, for example, use1-az1. Availability Zone names, for example, us-east-1a, don't map directly to Availability Zone IDs. For different AWS accounts, the same Availability Zone name might refer to a different Availability Zone ID.

For more information, see Regions and Availability Zones in the Amazon EC2 User Guide and AZ IDs for Your Resources in the AWS RAM User Guide.

Topics
• Component Support for Instances in US East (Ohio) us-east-2 (p. 800)
• Component Support for Instances in US East (N. Virginia) us-east-1 (p. 802)
• Component Support for Instances in US West (N. California) us-west-1 (p. 805)
• Component Support for Instances in US West (Oregon) us-west-2 (p. 807)
• Component Support for Instances in Asia Pacific (Hong Kong) ap-east-1 (p. 809)
• Component Support for Instances in Asia Pacific (Mumbai) ap-south-1 (p. 811)
• Component Support for Instances in Asia Pacific (Seoul) ap-northeast-2 (p. 814)
• Component Support for Instances in Asia Pacific (Singapore) ap-southeast-1 (p. 816)
• Component Support for Instances in Asia Pacific (Sydney) ap-southeast-2 (p. 818)
• Component Support for Instances in Asia Pacific (Tokyo) ap-northeast-1 (p. 820)
• Component Support for Instances in Canada (Central) ca-central-1 (p. 822)
• Component Support for Instances in EU (Frankfurt) eu-central-1 (p. 825)
• Component Support for Instances in EU (Ireland) eu-west-1 (p. 827)
- Component Support for Instances in EU (London) eu-west-2 (p. 829)
- Component Support for Instances in EU (Paris) eu-west-3 (p. 831)
- Component Support for Instances in EU (Stockholm) eu-north-1 (p. 833)
- Component Support for Instances in Middle East (Bahrain) me-south-1 (p. 835)
- Component Support for Instances in South America (Sao Paulo) sa-east-1 (p. 837)
- Component Support for Instances in AWS GovCloud (US-Gov-West) us-gov-west-1 (p. 840)

Component Support for Instances in US East (Ohio) us-east-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>use2-az1</th>
<th>use2-az2</th>
<th>use2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>use2-az1</td>
<td>use2-az2</td>
<td>use2-az3</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>use2-az1</td>
<td>use2-az2</td>
<td>use2-az3</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
</tbody>
</table>

Component Support for Instances in US East (N. Virginia) us-east-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>use1-az1</th>
<th>use1-az2</th>
<th>use1-az3</th>
<th>use1-az4</th>
<th>use1-az5</th>
<th>use1-az6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td></td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>Instance Type</td>
<td>use1-az1</td>
<td>use1-az2</td>
<td>use1-az3</td>
<td>use1-az4</td>
<td>use1-az5</td>
<td>use1-az6</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch</td>
<td>None</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>Notebook, Batch, Endpoint</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
<td>Notebook, Batch, Endpoint</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>Instance Type</td>
<td>use1-az1</td>
<td>use1-az2</td>
<td>use1-az3</td>
<td>use1-az4</td>
<td>use1-az5</td>
<td>use1-az6</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>None</td>
<td>Notebook, Training, Batch</td>
<td></td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>All</td>
<td>Training</td>
<td>None</td>
<td>None</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>All</td>
<td>Training</td>
<td>None</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>All</td>
<td>Training</td>
<td>None</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Training</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
</tbody>
</table>
## Component Support for Instances in US West (N. California) us-west-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>usw1-az1</th>
<th>usw1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>usw1-az1</td>
<td>usw1-az3</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
### Component Support for Instances in US West (Oregon) us-west-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>usw1-az1</th>
<th>usw1-az3</th>
<th>usw2-az2</th>
<th>usw2-az3</th>
<th>usw2-az4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>usw2-az1</th>
<th>usw2-az2</th>
<th>usw2-az3</th>
<th>usw2-az4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>usw2-az1</td>
<td>usw2-az2</td>
<td>usw2-az3</td>
<td>usw2-az4</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>All</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>All</td>
<td>Notebook, Training, Endpoint</td>
<td>All</td>
<td>None</td>
</tr>
</tbody>
</table>
### Component Support for Instances in Asia Pacific (Hong Kong) ap-east-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>ape1-az1</th>
<th>ape1-az2</th>
<th>ape1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>Instance Type</td>
<td>ape1-az1</td>
<td>ape1-az2</td>
<td>ape1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
</tbody>
</table>
## Component Support for Instances in Asia Pacific (Mumbai) ap-south-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>ape1-az1</th>
<th>ape1-az2</th>
<th>ape1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>aps1-az1</th>
<th>aps1-az2</th>
<th>aps1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>Instance Type</td>
<td>aps1-az1</td>
<td>aps1-az2</td>
<td>aps1-az3</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Batch</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>aps1-az1</td>
<td>aps1-az2</td>
<td>aps1-az3</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
## Component Support for Instances in Asia Pacific (Seoul) ap-northeast-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apne2-az1</th>
<th>apne2-az2</th>
<th>apne2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch,</td>
<td>None</td>
<td>Training, Batch,</td>
</tr>
<tr>
<td></td>
<td>Endpoint</td>
<td></td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>apne2-az1</td>
<td>apne2-az2</td>
<td>apne2-az3</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
</tbody>
</table>
## Component Support for Instances in Asia Pacific (Singapore) ap-southeast-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apse1-az1</th>
<th>apse1-az2</th>
<th>apse1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>apse1-az1</td>
<td>apse1-az2</td>
<td>apse1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
## Instance Type Support for Instances in Asia Pacific (Sydney) ap-southeast-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apse1-az1</th>
<th>apse1-az2</th>
<th>apse1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

## Component Support for Instances in Asia Pacific (Sydney) ap-southeast-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apse2-az1</th>
<th>apse2-az2</th>
<th>apse2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>apse2-az1</td>
<td>apse2-az2</td>
<td>apse2-az3</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>Notebook, Training, Batch</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Batch</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Batch</td>
<td>None</td>
</tr>
</tbody>
</table>
### Component Support for Instances in Asia Pacific (Tokyo) ap-northeast-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apse2-az1</th>
<th>apse2-az2</th>
<th>apse2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>apne1-az1</th>
<th>apne1-az2</th>
<th>apne1-az3</th>
<th>apne1-az4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>apne1-az1</td>
<td>apne1-az2</td>
<td>apne1-az3</td>
<td>apne1-az4</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>None</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>Notebook, Endpoint</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>None</td>
<td>Notebook</td>
</tr>
</tbody>
</table>
### Component Support for Instances in Canada (Central) ca-central-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>cac1-az1</th>
<th>cac1-az2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>Instance Type</td>
<td>cac1-az1</td>
<td>cac1-az2</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>cac1-az1</td>
<td>cac1-az2</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Component Support for Instances in EU (Frankfurt) eu-central-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euc1-az1</th>
<th>euc1-az2</th>
<th>euc1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>euc1-az1</td>
<td>euc1-az2</td>
<td>euc1-az3</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Training</td>
<td>Training, Batch</td>
<td>Training, Batch</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
### Instance Type | eu1-az1 | eu1-az2 | eu1-az3
---|---|---|---
ml.eia2.large | None | None | None
ml.eia2.xlarge | None | None | None

### Component Support for Instances in EU (Ireland) eu-west-1

| Instance Type | euw1-az1 | euw1-az2 | euw1-az3 |
---|---|---|---|
ml.t2.medium | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
ml.t2.large | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
ml.t2.xlarge | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
ml.t2.2xlarge | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
ml.t3.medium | Notebook | Notebook | Notebook |
ml.t3.large | Notebook | Notebook | Notebook |
ml.t3.xlarge | Notebook | Notebook | Notebook |
ml.t3.2xlarge | Notebook | Notebook | Notebook |
ml.m4.xlarge | All | All | All |
ml.m4.2xlarge | All | All | All |
ml.m4.4xlarge | All | Notebook, Training, Batch | All |
ml.m4.10xlarge | All | All | All |
ml.m4.16xlarge | All | All | All |
ml.m5.large | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
ml.m5.xlarge | All | All | All |
ml.m5.2xlarge | All | All | All |
ml.m5.4xlarge | All | All | All |
ml.m5.12xlarge | All | All | All |
ml.m5.24xlarge | All | All | All |
ml.r5.large | None | None | None |
ml.r5.xlarge | None | None | None |
ml.r5.2xlarge | None | None | None |
ml.r5.4xlarge | None | None | None |
<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euw1-az1</th>
<th>euw1-az2</th>
<th>euw1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>All</td>
<td>All</td>
<td>Notebook, Training, Batch</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
### Instance Type

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euw1-az1</th>
<th>euw1-az2</th>
<th>euw1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

### Component Support for Instances in EU (London) eu-west-2

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euw2-az1</th>
<th>euw2-az2</th>
<th>euw2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>euw2-az1</td>
<td>euw2-az2</td>
<td>euw2-az3</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
### Component Support for Instances in EU (Paris) eu-west-3

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euw2-az1</th>
<th>euw2-az2</th>
<th>euw2-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>euw3-az1</th>
<th>euw3-az2</th>
<th>euw3-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td><strong>Instance Type</strong></td>
<td>euw3-az1</td>
<td>euw3-az2</td>
<td>euw3-az3</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Component Support for Instances in EU (Stockholm) eu-north-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>eun1-az1</th>
<th>eun1-az2</th>
<th>eun1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>eun1-az1</td>
<td>eun1-az2</td>
<td>eun1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
## Component Support for Instances in Middle East (Bahrain) me-south-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>mes1-az1</th>
<th>mes1-az2</th>
<th>mes1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>Notebook</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td><code>mes1-az1</code></td>
<td><code>mes1-az2</code></td>
<td><code>mes1-az3</code></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
<td>Notebook, Training, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
</tbody>
</table>
### Component Support for Instances in South America (Sao Paulo) sa-east-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>sae1-az1</th>
<th>sae1-az2</th>
<th>sae1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
</tbody>
</table>

---

---

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>mes1-az1</th>
<th>mes1-az2</th>
<th>mes1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>Training</td>
<td>Training</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>sae1-az1</td>
<td>sae1-az2</td>
<td>sae1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Notebook</td>
<td>None</td>
<td>Notebook</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>All</td>
<td>Endpoint</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch,</td>
<td>None</td>
<td>Training, Batch,</td>
</tr>
<tr>
<td></td>
<td>Endpoint</td>
<td></td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>None</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>Instance Type</td>
<td>sae1-az1</td>
<td>sae1-az2</td>
<td>sae1-az3</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
## Component Support for Instances in AWS GovCloud (US-Gov-West) us-gov-west-1

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>usgw1-az1</th>
<th>usgw1-az2</th>
<th>usgw1-az3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.t2.medium</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.large</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t2.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.t3.medium</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.t3.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.t3.xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.t3.2xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.m4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.10xlarge</td>
<td>Notebook, Training, Batch</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
</tr>
<tr>
<td>ml.m4.16xlarge</td>
<td>Notebook, Training, Batch</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.large</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
<td>Training, Batch, Endpoint</td>
</tr>
<tr>
<td>ml.m5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.12xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.m5.24xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.r5.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.r5.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.c4.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>usgw1-az1</td>
<td>usgw1-az2</td>
<td>usgw1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ml.c4.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c4.8xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.large</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Endpoint</td>
</tr>
<tr>
<td>ml.c5.xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.2xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.4xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.9xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5.18xlarge</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>ml.c5d.xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.2xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.4xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.9xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.c5d.18xlarge</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
<td>Notebook, Endpoint</td>
</tr>
<tr>
<td>ml.p2.xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p2.8xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p2.16xlarge</td>
<td>Endpoint</td>
<td>Endpoint</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.2xlarge</td>
<td>Notebook, Batch, Endpoint</td>
<td>Notebook, Batch, Endpoint</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.8xlarge</td>
<td>Notebook, Batch, Endpoint</td>
<td>Notebook, Batch, Endpoint</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3.16xlarge</td>
<td>Notebook, Batch</td>
<td>Notebook, Batch</td>
<td>Training</td>
</tr>
<tr>
<td>ml.p3dn.24xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.2xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.4xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.8xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.12xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.g4dn.16xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia1.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Instance Type</td>
<td>usgw1-az1</td>
<td>usgw1-az2</td>
<td>usgw1-az3</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>ml.eia1.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.medium</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.large</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ml.eia2.xlarge</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
API Reference

This section contains the API Reference documentation.

Note
See Amazon Augmented AI Runtime for Amazon Augmented AI API Reference documentation.

Topics

- Actions (p. 843)
- Data Types (p. 1264)

Actions

The following actions are supported by Amazon SageMaker Service:

- AddTags (p. 850)
- AssociateTrialComponent (p. 852)
- CreateAlgorithm (p. 854)
- CreateApp (p. 859)
- CreateAutoMLJob (p. 862)
- CreateCodeRepository (p. 866)
- CreateCompilationJob (p. 868)
- CreateDomain (p. 871)
- CreateEndpoint (p. 875)
- CreateEndpointConfig (p. 878)
- CreateExperiment (p. 882)
- CreateFlowDefinition (p. 885)
- CreateHumanTaskUi (p. 888)
- CreateHyperParameterTuningJob (p. 890)
- CreateLabelingJob (p. 897)
- CreateModel (p. 902)
- CreateModelPackage (p. 906)
- CreateMonitoringSchedule (p. 910)
- CreateNotebookInstance (p. 913)
- CreateNotebookInstanceLifecycleConfig (p. 919)
- CreatePresignedDomainUrl (p. 922)
- CreatePresignedNotebookInstanceUrl (p. 924)
- CreateProcessingJob (p. 926)
- CreateTrainingJob (p. 931)
- CreateTransformJob (p. 939)
- CreateTrial (p. 944)
- CreateTrialComponent (p. 947)
- CreateUserProfile (p. 951)
- CreateWorkteam (p. 954)
• DeleteAlgorithm (p. 957)
• DeleteApp (p. 958)
• DeleteCodeRepository (p. 960)
• DeleteDomain (p. 961)
• DeleteEndpoint (p. 963)
• DeleteEndpointConfig (p. 965)
• DeleteExperiment (p. 966)
• DeleteFlowDefinition (p. 968)
• DeleteModel (p. 970)
• DeleteModelPackage (p. 971)
• DeleteMonitoringSchedule (p. 973)
• DeleteNotebookInstance (p. 975)
• DeleteNotebookInstanceLifecycleConfig (p. 977)
• DeleteTags (p. 978)
• DeleteTrial (p. 980)
• DeleteTrialComponent (p. 982)
• DeleteUserProfile (p. 984)
• DeleteWorkteam (p. 986)
• DescribeAlgorithm (p. 988)
• DescribeApp (p. 993)
• DescribeAutoMLJob (p. 997)
• DescribeCodeRepository (p. 1002)
• DescribeCompilationJob (p. 1004)
• DescribeDomain (p. 1008)
• DescribeEndpoint (p. 1012)
• DescribeEndpointConfig (p. 1015)
• DescribeExperiment (p. 1018)
• DescribeFlowDefinition (p. 1021)
• DescribeHumanTaskUi (p. 1024)
• DescribeHyperParameterTuningJob (p. 1026)
• DescribeLabelingJob (p. 1034)
• DescribeModel (p. 1040)
• DescribeModelPackage (p. 1043)
• DescribeMonitoringSchedule (p. 1047)
• DescribeNotebookInstance (p. 1051)
• DescribeNotebookInstanceLifecycleConfig (p. 1056)
• DescribeProcessingJob (p. 1059)
• DescribeSubscribedWorkteam (p. 1064)
• DescribeTrainingJob (p. 1066)
• DescribeTransformJob (p. 1075)
• DescribeTrial (p. 1080)
• DescribeTrialComponent (p. 1083)
• DescribeUserProfile (p. 1087)
• DescribeWorkteam (p. 1091)
• DisassociateTrialComponent (p. 1093)
• GetSearchSuggestions (p. 1095)
• ListAlgorithms (p. 1097)
• ListApps (p. 1100)
• ListAutoMLJobs (p. 1103)
• ListCandidatesForAutoMLJob (p. 1106)
• ListCodeRepositories (p. 1109)
• ListCompilationJobs (p. 1112)
• ListDomains (p. 1116)
• ListEndpointConfigs (p. 1118)
• ListEndpoints (p. 1121)
• ListExperiments (p. 1124)
• ListFlowDefinitions (p. 1127)
• ListHumanTaskUi (p. 1130)
• ListHyperParameterTuningJobs (p. 1133)
• ListLabelingJobs (p. 1137)
• ListLabelingJobsForWorkteam (p. 1141)
• ListModelPackages (p. 1144)
• ListModels (p. 1147)
• ListMonitoringExecutions (p. 1150)
• ListMonitoringSchedules (p. 1154)
• ListNotebookInstanceLifecycleConfigs (p. 1158)
• ListNotebookInstances (p. 1161)
• ListProcessingJobs (p. 1165)
• ListSubscribedWorkteams (p. 1168)
• ListTags (p. 1170)
• ListTrainingJobs (p. 1172)
• ListTrainingJobsForHyperParameterTuningJob (p. 1175)
• ListTransformJobs (p. 1178)
• ListTrialComponents (p. 1181)
• ListTrials (p. 1185)
• ListUserProfiles (p. 1188)
• ListWorkteams (p. 1191)
• RenderUiTemplate (p. 1194)
• Search (p. 1196)
• StartMonitoringSchedule (p. 1206)
• StartNotebookInstance (p. 1208)
• StopAutoMLJob (p. 1210)
• StopCompilationJob (p. 1212)
• StopHyperParameterTuningJob (p. 1214)
• StopLabelingJob (p. 1216)
• StopMonitoringSchedule (p. 1218)
• StopNotebookInstance (p. 1220)
• StopProcessingJob (p. 1222)
• StopTrainingJob (p. 1224)
• StopTransformJob (p. 1226)
• UpdateCodeRepository (p. 1228)
• UpdateDomain (p. 1230)
• UpdateEndpoint (p. 1233)
• UpdateEndpointWeightsAndCapacities (p. 1235)
• UpdateExperiment (p. 1237)
• UpdateMonitoringSchedule (p. 1239)
• UpdateNotebookInstance (p. 1242)
• UpdateNotebookInstanceLifecycleConfig (p. 1246)
• UpdateTrial (p. 1248)
• UpdateTrialComponent (p. 1250)
• UpdateUserProfile (p. 1254)
• UpdateWorkteam (p. 1257)

The following actions are supported by Amazon SageMaker Runtime:
• InvokeEndpoint (p. 1260)

Amazon SageMaker Service

The following actions are supported by Amazon SageMaker Service:
• AddTags (p. 850)
• AssociateTrialComponent (p. 852)
• CreateAlgorithm (p. 854)
• CreateApp (p. 859)
• CreateAutoMLJob (p. 862)
• CreateCodeRepository (p. 866)
• CreateCompilationJob (p. 868)
• CreateDomain (p. 871)
• CreateEndpoint (p. 875)
• CreateEndpointConfig (p. 878)
• CreateExperiment (p. 882)
• CreateFlowDefinition (p. 885)
• CreateHumanTaskUi (p. 888)
• CreateHyperParameterTuningJob (p. 890)
• CreateLabelingJob (p. 897)
• CreateModel (p. 902)
• CreateModelPackage (p. 906)
• CreateMonitoringSchedule (p. 910)
• CreateNotebookInstance (p. 913)
• CreateNotebookInstanceLifecycleConfig (p. 919)
• CreatePresignedDomainUrl (p. 922)
• CreatePresignedNotebookInstanceUrl (p. 924)
• CreateProcessingJob (p. 926)
• CreateTrainingJob (p. 931)
• CreateTransformJob (p. 939)
• CreateTrial (p. 944)
• CreateTrialComponent (p. 947)
• CreateUserProfile (p. 951)
• CreateWorkteam (p. 954)
• DeleteAlgorithm (p. 957)
• DeleteApp (p. 958)
• DeleteCodeRepository (p. 960)
• DeleteDomain (p. 961)
• DeleteEndpoint (p. 963)
• DeleteEndpointConfig (p. 965)
• DeleteExperiment (p. 966)
• DeleteFlowDefinition (p. 968)
• DeleteModel (p. 970)
• DeleteModelPackage (p. 971)
• DeleteMonitoringSchedule (p. 973)
• DeleteNotebookInstance (p. 975)
• DeleteNotebookInstanceLifecycleConfig (p. 977)
• DeleteTags (p. 978)
• DeleteTrial (p. 980)
• DeleteTrialComponent (p. 982)
• DeleteUserProfile (p. 984)
• DeleteWorkteam (p. 986)
• DescribeAlgorithm (p. 988)
• DescribeApp (p. 993)
• DescribeAutoMLJob (p. 997)
• DescribeCodeRepository (p. 1002)
• DescribeCompilationJob (p. 1004)
• DescribeDomain (p. 1008)
• DescribeEndpoint (p. 1012)
• DescribeEndpointConfig (p. 1015)
• DescribeExperiment (p. 1018)
• DescribeFlowDefinition (p. 1021)
• DescribeHumanTaskUi (p. 1024)
• DescribeHyperParameterTuningJob (p. 1026)
• DescribeLabelingJob (p. 1034)
• DescribeModel (p. 1040)
• DescribeModelPackage (p. 1043)
• DescribeMonitoringSchedule (p. 1047)
• DescribeNotebookInstance (p. 1051)
• DescribeNotebookInstanceLifecycleConfig (p. 1056)
• DescribeProcessingJob (p. 1059)
• DescribeSubscribedWorkteam (p. 1064)
• DescribeTrainingJob (p. 1066)
• DescribeTransformJob (p. 1075)
• DescribeTrial (p. 1080)
• DescribeTrialComponent (p. 1083)
• DescribeUserProfile (p. 1087)
• DescribeWorkteam (p. 1091)
• DisassociateTrialComponent (p. 1093)
• GetSearchSuggestions (p. 1095)
• ListAlgorithms (p. 1097)
• ListApps (p. 1100)
• ListAutoMLJobs (p. 1103)
• ListCandidatesForAutoMLJob (p. 1106)
• ListCodeRepositories (p. 1109)
• ListCompilationJobs (p. 1112)
• ListDomains (p. 1116)
• ListEndpointConfigs (p. 1118)
• ListEndpoints (p. 1121)
• ListExperiments (p. 1124)
• ListFlowDefinitions (p. 1127)
• ListHumanTaskUis (p. 1130)
• ListHyperParameterTuningJobs (p. 1133)
• ListLabelingJobs (p. 1137)
• ListLabelingJobsForWorkteam (p. 1141)
• ListModelPackages (p. 1144)
• ListModels (p. 1147)
• ListMonitoringExecutions (p. 1150)
• ListMonitoringSchedules (p. 1154)
• ListNotebookInstanceLifecycleConfigs (p. 1158)
• ListNotebookInstances (p. 1161)
• ListProcessingJobs (p. 1165)
• ListSubscribedWorkteams (p. 1168)
• ListTags (p. 1170)
• ListTrainingJobs (p. 1172)
• ListTrainingJobsForHyperParameterTuningJob (p. 1175)
• ListTransformJobs (p. 1178)
• ListTrialComponents (p. 1181)
• ListTrials (p. 1185)
• ListUserProfiles (p. 1188)
• ListWorkteams (p. 1191)
• RenderUiTemplate (p. 1194)
• Search (p. 1196)
• StartMonitoringSchedule (p. 1206)
• StartNotebookInstance (p. 1208)
• StopAutoMLJob (p. 1210)
• StopCompilationJob (p. 1212)
• StopHyperParameterTuningJob (p. 1214)
• StopLabelingJob (p. 1216)
• StopMonitoringSchedule (p. 1218)
• StopNotebookInstance (p. 1220)
• StopProcessingJob (p. 1222)
• StopTrainingJob (p. 1224)
• StopTransformJob (p. 1226)
- UpdateCodeRepository (p. 1228)
- UpdateDomain (p. 1230)
- UpdateEndpoint (p. 1233)
- UpdateEndpointWeightsAndCapacities (p. 1235)
- UpdateExperiment (p. 1237)
- UpdateMonitoringSchedule (p. 1239)
- UpdateNotebookInstance (p. 1242)
- UpdateNotebookInstanceLifecycleConfig (p. 1246)
- UpdateTrial (p. 1248)
- UpdateTrialComponent (p. 1250)
- UpdateUserProfile (p. 1254)
- UpdateWorkteam (p. 1257)
AddTags
Service: Amazon SageMaker Service

Adds or overwrites one or more tags for the specified Amazon SageMaker resource. You can add tags to notebook instances, training jobs, hyperparameter tuning jobs, batch transform jobs, models, labeling jobs, work teams, endpoint configurations, and endpoints.

Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see AWS Tagging Strategies.

Note
Tags that you add to a hyperparameter tuning job by calling this API are also added to any training jobs that the hyperparameter tuning job launches after you call this API, but not to training jobs that the hyperparameter tuning job launched before you called this API. To make sure that the tags associated with a hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches, add the tags when you first create the tuning job by specifying them in the Tags parameter of CreateHyperParameterTuningJob (p. 890)

Request Syntax

```
{
   "ResourceArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

ResourceArn (p. 850)

The Amazon Resource Name (ARN) of the resource that you want to tag.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:.*

Required: Yes

Tags (p. 850)

An array of Tag objects. Each tag is a key-value pair. Only the key parameter is required. If you don't specify a value, Amazon SageMaker sets the value to an empty string.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: Yes
Response Syntax

```json
{
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

Tags (p. 851)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**AssociateTrialComponent**  
Service: Amazon SageMaker Service

Associates a trial component with a trial. A trial component can be associated with multiple trials. To disassociate a trial component from a trial, call the DisassociateTrialComponent (p. 1093) API.

**Request Syntax**

```json
{
  "TrialComponentName": "string",
  "TrialName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialComponentName (p. 852)**

The name of the component to associated with the trial.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**TrialName (p. 852)**

The name of the trial to associate with.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**Response Syntax**

```json
{
  "TrialArn": "string",
  "TrialComponentArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialArn (p. 852)**

The Amazon Resource Name (ARN) of the trial.
Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[\w\-]*:sagemaker:[\w\-]*:[0-9]{12}:experiment-trial/.*

**TrialComponentArn (p. 852)**

The ARN of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[\w\-]*:sagemaker:[\w\-]*:[0-9]{12}:experiment-trial-component/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateAlgorithm
Service: Amazon SageMaker Service

Create a machine learning algorithm that you can use in Amazon SageMaker and list in the AWS Marketplace.

Request Syntax

```json
{
  "AlgorithmDescription": "string",
  "AlgorithmName": "string",
  "CertifyForMarketplace": boolean,
  "InferenceSpecification": {
    "Containers": [
      {
        "ContainerHostname": "string",
        "Image": "string",
        "ImageDigest": "string",
        "ModelDataUrl": "string",
        "ProductID": "string"
      }
    ],
    "SupportedContentTypes": [ "string" ],
    "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
    "SupportedResponseMIMETypes": [ "string" ],
    "SupportedTransformInstanceTypes": [ "string" ]
  },
  "TrainingSpecification": {
    "MetricDefinitions": [
      {
        "Name": "string",
        "Regex": "string"
      }
    ],
    "SupportedHyperParameters": [
      {
        "DefaultValue": "string",
        "Description": "string",
        "IsRequired": boolean,
        "IsTunable": boolean,
        "Name": "string",
        "Range": {
          "CategoricalParameterRangeSpecification": {
            "Values": [ "string" ]
          },
          "ContinuousParameterRangeSpecification": {
            "MaxValue": "string",
            "MinValue": "string"
          },
          "IntegerParameterRangeSpecification": {
            "MaxValue": "string",
            "MinValue": "string"
          }
        },
        "Type": "string"
      }
    ],
    "SupportedTrainingInstanceTypes": [ "string" ],
    "SupportedTuningJobObjectiveMetrics": [ 
      {
        "MetricName": "string",
        "Type": "string"
      }
    ]
  }
}
```
"SupportsDistributedTraining": boolean,
"TrainingChannels": [
  
  "Description": "string",
  "IsRequired": boolean,
  "Name": "string",
  "SupportedCompressionTypes": [ "string" ],
  "SupportedContentTypes": [ "string" ],
  "SupportedInputModes": [ "string" ]
]
],
"TrainingImage": "string",
"TrainingImageDigest": "string"
},
"ValidationSpecification": {
  "ValidationProfiles": [
    
    "ProfileName": "string",
    "TrainingJobDefinition": {
      "HyperParameters": {
        "string": "string"
      },
      "InputDataConfig": [
        
        "ChannelName": "string",
        "CompressionType": "string",
        "ContentType": "string",
        "DataSource": {
          "FileSystemDataSource": {
            "DirectoryPath": "string",
            "FileSystemAccessMode": "string",
            "FileSystemId": "string",
            "FileSystemType": "string"
          },
          "S3DataSource": {
            "AttributeNames": [ "string" ],
            "S3DataDistributionType": "string",
            "S3DataType": "string",
            "S3Uri": "string"
          }
        },
        "InputMode": "string",
        "RecordWrapperType": "string",
        "ShuffleConfig": {
          "Seed": number
        }
      }
    ],
    "OutputDataConfig": {
      "KmsKeyId": "string",
      "S3OutputPath": "string"
    },
    "ResourceConfig": {
      "InstanceCount": number,
      "InstanceType": "string",
      "VolumeKmsKeyId": "string",
      "VolumeSizeInGB": number
    },
    "StoppingCondition": {
      "MaxRuntimeInSeconds": number,
      "MaxWaitTimeInSeconds": number
    },
    "TrainingInputMode": "string"
  ],
  "TransformJobDefinition": {
    "BatchStrategy": "string",
    "TransformImage": "string",
    "TransformImageDigest": "string"
  }
}
"Environment": {  
   "string": "string"
},
"MaxConcurrentTransforms": number,
"MaxPayloadInMB": number,
"TransformInput": {  
   "CompressionType": "string",
   "ContentType": "string",
   "DataSource": {  
      "S3DataSource": {  
         "S3DataType": "string",
         "S3Uri": "string"
      }
   },
   "SplitType": "string"
},
"TransformOutput": {  
   "Accept": "string",
   "AssembleWith": "string",
   "KmsKeyId": "string",
   "S3OutputPath": "string"
},
"TransformResources": {  
   "InstanceCount": number,
   "InstanceType": "string",
   "VolumeKmsKeyId": "string"
}
},
"ValidationRole": "string"
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AlgorithmDescription (p. 854)**

A description of the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

**AlgorithmName (p. 854)**

The name of the algorithm.

Type: String


Pattern: ^[a-zA-Z0-9](\*[a-zA-Z0-9])*$

Required: Yes
CertifyForMarketplace (p. 854)

Whether to certify the algorithm so that it can be listed in AWS Marketplace.

Type: Boolean

Required: No

InferenceSpecification (p. 854)

Specifies details about inference jobs that the algorithm runs, including the following:
- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the algorithm supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the algorithm supports for inference.

Type: InferenceSpecification (p. 1396) object

Required: No

TrainingSpecification (p. 854)

Specifies details about training jobs run by this algorithm, including the following:
- The Amazon ECR path of the container and the version digest of the algorithm.
- The hyperparameters that the algorithm supports.
- The instance types that the algorithm supports for training.
- Whether the algorithm supports distributed training.
- The metrics that the algorithm emits to Amazon CloudWatch.
- Which metrics that the algorithm emits can be used as the objective metric for hyperparameter tuning jobs.
- The input channels that the algorithm supports for training data. For example, an algorithm might support train, validation, and test channels.

Type: TrainingSpecification (p. 1533) object

Required: Yes

ValidationSpecification (p. 854)

Specifies configurations for one or more training jobs and that Amazon SageMaker runs to test the algorithm's training code and, optionally, one or more batch transform jobs that Amazon SageMaker runs to test the algorithm's inference code.

Type: AlgorithmValidationSpecification (p. 1281) object

Required: No

Response Syntax

```
{
    "AlgorithmArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
AlgorithmArn (p. 857)

The Amazon Resource Name (ARN) of the new algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateApp  
Service: Amazon SageMaker Service

Creates a running App for the specified UserProfile. Supported Apps are JupyterServer and KernelGateway. This operation is automatically invoked by Amazon SageMaker Amazon SageMaker Studio (Studio) upon access to the associated Studio Domain, and when new kernel configurations are selected by the user. A user may have multiple Apps active simultaneously. Apps will automatically terminate and be deleted when stopped from within Studio, or when the DeleteApp API is manually called. UserProfiles are limited to 5 concurrently running Apps at a time.

Request Syntax

```
{
    "AppName": "string",
    "AppType": "string",
    "DomainId": "string",
    "ResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AppName (p. 859)

   The name of the app.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]$  

Required: Yes

AppType (p. 859)

   The type of app.

Type: String

Valid Values: JupyterServer | KernelGateway | TensorBoard

Required: Yes

DomainId (p. 859)

   The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: Yes

ResourceSpec (p. 859)
The instance type and quantity.
Type: ResourceSpec (p. 1499) object
Required: No

Tags (p. 859)
Each tag consists of a key and an optional value. Tag keys must be unique per resource.
Type: Array of Tag (p. 1517) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

UserProfileName (p. 859)
The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
Required: Yes

Response Syntax

```json
{
   "AppArn": "string"
}
```

Response Elements
If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

AppArn (p. 860)
The app's Amazon Resource Name (ARN).
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-zA-Z]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:app/.*

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateAutoMLJob
Service: Amazon SageMaker Service

Creates an AutoPilot job.

Request Syntax

```json
{
    "AutoMLJobConfig": {
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": number,
            "MaxCandidates": number,
            "MaxRuntimePerTrainingJobInSeconds": number
        },
        "SecurityConfig": {
            "EnableInterContainerTrafficEncryption": boolean,
            "VolumeKmsKeyId": "string",
            "VpcConfig": {
                "SecurityGroupIds": [ "string" ],
                "Subnets": [ "string" ]
            }
        },
        "AutoMLJobName": "string",
        "AutoMLJobObjective": {
            "MetricName": "string"
        },
        "GenerateCandidateDefinitionsOnly": boolean,
        "InputDataConfig": [
            {
                "CompressionType": "string",
                "DataSource": {
                    "S3DataSource": {
                        "S3DataType": "string",
                        "S3Uri": "string"
                    }
                },
                "TargetAttributeName": "string"
            }
        ],
        "OutputDataConfig": {
            "KmsKeyId": "string",
            "S3OutputPath": "string"
        },
        "ProblemType": "string",
        "RoleArn": "string",
        "Tags": [
            {
                "Key": "string",
                "Value": "string"
            }
        ]
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.
AutoMLJobConfig (p. 862)
Contains CompletionCriteria and SecurityConfig.
Type: AutoMLJobConfig (p. 1299) object
Required: No

AutoMLJobName (p. 862)
Identifies an AutoPilot job. Must be unique to your account and is case-insensitive.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: Yes

AutoMLJobObjective (p. 862)
Defines the job's objective. You provide a MetricName and AutoML will infer minimize or maximize. If this is not provided, the most commonly used ObjectiveMetric for problem type will be selected.
Type: AutoMLJobObjective (p. 1300) object
Required: No

GenerateCandidateDefinitionsOnly (p. 862)
This will generate possible candidates without training a model. A candidate is a combination of data preprocessors, algorithms, and algorithm parameter settings.
Type: Boolean
Required: No

InputDataConfig (p. 862)
Similar to InputDataConfig supported by Tuning. Format(s) supported: CSV.
Type: Array of AutoMLChannel (p. 1294) objects
Array Members: Minimum number of 1 item. Maximum number of 20 items.
Required: Yes

OutputDataConfig (p. 862)
Similar to OutputDataConfig supported by Tuning. Format(s) supported: CSV.
Type: AutoMLOutputDataConfig (p. 1303) object
Required: Yes

ProblemType (p. 862)
Defines the kind of preprocessing and algorithms intended for the candidates. Options include: BinaryClassification, MulticlassClassification, and Regression.
Type: String
Valid Values: BinaryClassification | MulticlassClassification | Regression
Required: No
RoleArn (p. 862)

The ARN of the role that will be used to access the data.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9=\+,\.@\-_]+$

Required: Yes

Tags (p. 862)

Each tag consists of a key and an optional value. Tag keys must be unique per resource.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

Response Syntax

```
{
   "AutoMLJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AutoMLJobArn (p. 864)

When a job is created, it is assigned a unique ARN.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:automl-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateCodeRepository
Service: Amazon SageMaker Service

Creates a Git repository as a resource in your Amazon SageMaker account. You can associate the repository with notebook instances so that you can use Git source control for the notebooks you create. The Git repository is a resource in your Amazon SageMaker account, so it can be associated with more than one notebook instance, and it persists independently from the lifecycle of any notebook instances it is associated with.

The repository can be hosted either in AWS CodeCommit or in any other Git repository.

Request Syntax

```
{
    "CodeRepositoryName": "string",
    "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 866)**

The name of the Git repository. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

- Type: String
- Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]$*
- Required: Yes

**GitConfig (p. 866)**

Specifies details about the repository, including the URL where the repository is located, the default branch, and credentials to use to access the repository.

- Type: GitConfig (p. 1362) object
- Required: Yes

Response Syntax

```
{
    "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CodeRepositoryArn (p. 866)**

The Amazon Resource Name (ARN) of the new repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateCompilationJob
Service: Amazon SageMaker Service

Starts a model compilation job. After the model has been compiled, Amazon SageMaker saves the resulting model artifacts to an Amazon Simple Storage Service (Amazon S3) bucket that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts with AWS IoT Greengrass. In that case, deploy them as an ML resource.

In the request body, you provide the following:

- A name for the compilation job
- Information about the input model artifacts
- The output location for the compiled model and the device (target) that the model runs on
- The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker assumes to perform the model compilation job

You can also provide a Tag to track the model compilation job's resource use and costs. The response body contains the CompilationJobArn for the compiled job.

To stop a model compilation job, use StopCompilationJob (p. 1212). To get information about a particular model compilation job, use DescribeCompilationJob (p. 1004). To get information about multiple model compilation jobs, use ListCompilationJobs (p. 1112).

Request Syntax

```
{
    "CompilationJobName": "string",
    "InputConfig": {
        "DataInputConfig": "string",
        "Framework": "string",
        "S3Uri": "string"
    },
    "OutputConfig": {
        "S3OutputLocation": "string",
        "TargetDevice": "string"
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number,
        "MaxWaitTimeInSeconds": number
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CompilationJobName (p. 868)

A name for the model compilation job. The name must be unique within the AWS Region and within your AWS account.

Type: String

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

InputConfig (p. 868)

Provides information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: `InputConfig (p. 1398) object`

Required: Yes

OutputConfig (p. 868)

Provides information about the output location for the compiled model and the target device the model runs on.

Type: `OutputConfig (p. 1465) object`

Required: Yes

RoleArn (p. 868)

The Amazon Resource Name (ARN) of an IAM role that enables Amazon SageMaker to perform tasks on your behalf.

During model compilation, Amazon SageMaker needs your permission to:
- Read input data from an S3 bucket
- Write model artifacts to an S3 bucket
- Write logs to Amazon CloudWatch Logs
- Publish metrics to Amazon CloudWatch

You grant permissions for all of these tasks to an IAM role. To pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission. For more information, see Amazon SageMaker Roles.

Type: String


Pattern: `^arn:aws[a-z\-]+:iam::\d{12}:role/?[a-zA-Z\-0-9+=,.@\-_\/]+$`

Required: Yes

StoppingCondition (p. 868)

Specifies a limit to how long a model compilation job can run. When the job reaches the time limit, Amazon SageMaker ends the compilation job. Use this API to cap model training costs.

Type: `StoppingCondition (p. 1513) object`

Required: Yes

Response Syntax

```json
{
  "CompilationJobArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

CompilationJobArn (p. 869)
If the action is successful, the service sends back an HTTP 200 response. Amazon SageMaker returns the following data in JSON format:

- **CompilationJobArn**: The Amazon Resource Name (ARN) of the compiled job.

  Type: String

  Length Constraints: Maximum length of 256.

  Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateDomain
Service: Amazon SageMaker Service

Creates a Domain for Amazon SageMaker Amazon SageMaker Studio (Studio), which can be accessed by end-users in a web browser. A Domain has an associated directory, list of authorized users, and a variety of security, application, policies, and Amazon Virtual Private Cloud configurations. An AWS account is limited to one Domain, per region. Users within a domain can share notebook files and other artifacts with each other. When a Domain is created, an Amazon Elastic File System (EFS) is also created for use by all of the users within the Domain. Each user receives a private home directory within the EFS for notebooks, Git repositories, and data files.

Request Syntax

```json
{
  "AuthMode": "string",
  "DefaultUserSettings": {
    "ExecutionRole": "string",
    "JupyterServerAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "KernelGatewayAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "SecurityGroups": [ "string" ],
    "SharingSettings": {
      "NotebookOutputOption": "string",
      "S3KmsKeyId": "string",
      "S3OutputPath": "string"
    },
    "TensorBoardAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    }
  },
  "DomainName": "string",
  "HomeEfsFileSystemKmsKeyId": "string",
  "SubnetIds": [ "string" ],
  "Tags": [ {
    "Key": "string",
    "Value": "string"
  } ],
  "VpcId": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.
**AuthMode (p. 871)**

The mode of authentication that member use to access the domain.

Type: String

Valid Values: SSO | IAM

Required: Yes

**DefaultUserSettings (p. 871)**

The default user settings.

Type: UserSettings (p. 1575) object

Required: Yes

**DomainName (p. 871)**

A name for the domain.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9\-]*[a-zA-Z0-9\-]*$`

Required: Yes

**HomeEfsFileSystemKmsKeyId (p. 871)**

The AWS Key Management Service encryption key ID.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.\*`

Required: No

**SubnetIds (p. 871)**

Security setting to limit to a set of subnets.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 16 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: Yes

**Tags (p. 871)**

Each tag consists of a key and an optional value. Tag keys must be unique per resource.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No
VpcId (p. 871)

Security setting to limit the domain's communication to a Amazon Virtual Private Cloud.

Type: String
Length Constraints: Maximum length of 32.
Pattern: \[-0-9a-zA-Z\]+
Required: Yes

Response Syntax

```json
{
    "DomainArn": "string",
    "Url": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DomainArn (p. 873)

The Amazon Resource Name (ARN) of the created domain.

Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws\[a-z\-\]*:sagemaker:\[a-z0-9\-\]*:[0-9]{12}:domain/.*

Url (p. 873)

The URL to the created domain.

Type: String
Length Constraints: Maximum length of 1024.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateEndpoint
Service: Amazon SageMaker Service

Creates an endpoint using the endpoint configuration specified in the request. Amazon SageMaker uses the endpoint to provision resources and deploy models. You create the endpoint configuration with the CreateEndpointConfig API.

**Note**
Use this API only for hosting models using Amazon SageMaker hosting services. You must not delete an EndpointConfig in use by an endpoint that is live or while the UpdateEndpoint or CreateEndpoint operations are being performed on the endpoint. To update an endpoint, you must create a new EndpointConfig.

The endpoint name must be unique within an AWS Region in your AWS account.

When it receives the request, Amazon SageMaker creates the endpoint, launches the resources (ML compute instances), and deploys the model(s) on them.

When Amazon SageMaker receives the request, it sets the endpoint status to **Creating**. After it creates the endpoint, it sets the status to **InService**. Amazon SageMaker can then process incoming requests for inferences. To check the status of an endpoint, use the DescribeEndpoint API.

For an example, see Exercise 1: Using the K-Means Algorithm Provided by Amazon SageMaker.

If any of the models hosted at this endpoint get model data from an Amazon S3 location, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provided. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS in an AWS Region in the AWS Identity and Access Management User Guide.

**Request Syntax**

```json
{
  "EndpointConfigName": "string",
  "EndpointName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 875)**

The name of an endpoint configuration. For more information, see CreateEndpointConfig.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](~*[a-zA-Z0-9])*$`

Required: Yes
**EndpointName (p. 875)**

The name of the endpoint. The name must be unique within an AWS Region in your AWS account.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**Tags (p. 875)**

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**Response Syntax**

```json
{
   "EndpointArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**EndpointArn (p. 876)**

The Amazon Resource Name (ARN) of the endpoint.

Type: String


Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateEndpointConfig
Service: Amazon SageMaker Service

Creates an endpoint configuration that Amazon SageMaker hosting services uses to deploy models. In the configuration, you identify one or more models, created using the CreateModel API, to deploy and the resources that you want Amazon SageMaker to provision. Then you call the CreateEndpoint API.

**Note**
Use this API only if you want to use Amazon SageMaker hosting services to deploy models into production.

In the request, you define one or more ProductionVariants, each of which identifies a model. Each ProductionVariant parameter also describes the resources that you want Amazon SageMaker to provision. This includes the number and type of ML compute instances to deploy.

If you are hosting multiple models, you also assign a VariantWeight to specify how much traffic you want to allocate to each model. For example, suppose that you want to host two models, A and B, and you assign traffic weight 2 for model A and 1 for model B. Amazon SageMaker distributes two-thirds of the traffic to Model A, and one-third to model B.

**Request Syntax**

```json
{
  "DataCaptureConfig": {
    "CaptureContentTypeHeader": {
      "CsvContentTypes": ["string"],
      "JsonContentTypes": ["string"]
    },
    "CaptureOptions": [
      {
        "CaptureMode": "string"
      }
    ],
    "DestinationS3Uri": "string",
    "EnableCapture": boolean,
    "InitialSamplingPercentage": number,
    "KmsKeyId": "string"
  },
  "EndpointConfigName": "string",
  "KmsKeyId": "string",
  "ProductionVariants": [
    {
      "AcceleratorType": "string",
      "InitialInstanceCount": number,
      "InitialVariantWeight": number,
      "InstanceType": "string",
      "ModelName": "string",
      "VariantName": "string"
    }
  ],
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).
The request accepts the following data in JSON format.

**DataCaptureConfig (p. 878)**

Type: DataCaptureConfig (p. 1326) object

Required: No

**EndpointConfigName (p. 878)**

The name of the endpoint configuration. You specify this name in a CreateEndpoint request.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**KmsKeyId (p. 878)**

The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance that hosts the endpoint.

The KmsKeyId can be any of the following formats:

- Key ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- Key ARN: arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- Alias name: alias/ExampleAlias

The KMS key policy must grant permission to the IAM role that you specify in your CreateEndpoint, UpdateEndpoint requests. For more information, refer to the AWS Key Management Service section Using Key Policies in AWS KMS

**Note**

Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a KmsKeyId when using an instance type with local storage. If any of the models that you specify in the ProductionVariants parameter use nitro-based instances with local storage, do not specify a value for the KmsKeyId parameter. If you specify a value for KmsKeyId when using any nitro-based instances with local storage, the call to CreateEndpointConfig fails.

For a list of instance types that support local instance storage, see Instance Store Volumes. For more information about local instance storage encryption, see SSD Instance Store Volumes.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

**ProductionVariants (p. 878)**

An list of ProductionVariant objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 1484) objects
Array Members: Minimum number of 1 item. Maximum number of 10 items.
Required: Yes

**Tags (p. 878)**

A list of key-value pairs. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Type: Array of Tag (p. 1517) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**Response Syntax**

```json
{
    "EndpointConfigArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**EndpointConfigArn (p. 880)**

The Amazon Resource Name (ARN) of the endpoint configuration.
Type: String
Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*`

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 1579)].

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateExperiment
Service: Amazon SageMaker Service

Creates an Amazon SageMaker experiment. An experiment is a collection of trials that are observed, compared and evaluated as a group. A trial is a set of steps, called trial components, that produce a machine learning model.

The goal of an experiment is to determine the components that produce the best model. Multiple trials are performed, each one isolating and measuring the impact of a change to one or more inputs, while keeping the remaining inputs constant.

When you use Amazon SageMaker Studio or the Amazon SageMaker Python SDK, all experiments, trials, and trial components are automatically tracked, logged, and indexed. When you use the AWS SDK for Python (Boto), you must use the logging APIs provided by the SDK.

You can add tags to experiments, trials, trial components and then use the Search (p. 1196) API to search for the tags.

To add a description to an experiment, specify the optional Description parameter. To add a description later, or to change the description, call the UpdateExperiment (p. 1237) API.

To get a list of all your experiments, call the ListExperiments (p. 1124) API. To view an experiment's properties, call the DescribeExperiment (p. 1018) API. To get a list of all the trials associated with an experiment, call the ListTrials (p. 1185) API. To create a trial call the CreateTrial (p. 944) API.

Request Syntax

```json
{
  "Description": "string",
  "DisplayName": "string",
  "ExperimentName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**Description (p. 882)**

The description of the experiment.

Type: String

Length Constraints: Maximum length of 3072.

Pattern: .*

Required: No

**DisplayName (p. 882)**

The name of the experiment as displayed. The name doesn't need to be unique. If you don't specify DisplayName, the value in ExperimentName is displayed.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: No

**ExperimentName (p. 882)**

The name of the experiment. The name must be unique in your AWS account and is not case-sensitive.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: Yes

**Tags (p. 882)**

A list of tags to associate with the experiment. You can use Search (p. 1196) API to search on the tags.
Type: Array of Tag (p. 1517) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**Response Syntax**

```json
{
   "ExperimentArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ExperimentArn (p. 883)**

The Amazon Resource Name (ARN) of the experiment.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-\*:0-9]{12}:experiment/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateFlowDefinition
Service: Amazon SageMaker Service

Creates a flow definition.

Request Syntax

```json
{
  "FlowDefinitionName": "string",
  "HumanLoopActivationConfig": {
    "HumanLoopActivationConditionsConfig": {
      "HumanLoopActivationConditions": "string"
    },
    "HumanLoopRequestSource": {
      "AwsManagedHumanLoopRequestSource": "string"
    }
  },
  "HumanLoopConfig": {
    "HumanTaskUiArn": "string",
    "PublicWorkforceTaskPrice": {
      "AmountInUsd": {
        "Cents": number,
        "Dollars": number,
        "TenthFractionsOfACent": number
      }
    },
    "TaskAvailabilityLifetimeInSeconds": number,
    "TaskCount": number,
    "TaskDescription": "string",
    "TaskKeywords": [ "string" ],
    "TaskTimeLimitInSeconds": number,
    "TaskTitle": "string",
    "WorkteamArn": "string"
  },
  "OutputConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "RoleArn": "string",
  "Tags": [ {
    "Key": "string",
    "Value": "string"
  } ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**FlowDefinitionName (p. 885)**

The name of your flow definition.

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

HumanLoopActivationConfig (p. 885)
An object containing information about the events that trigger a human workflow.
Type: HumanLoopActivationConfig (p. 1365) object  
Required: No

HumanLoopConfig (p. 885)
An object containing information about the tasks the human reviewers will perform.
Type: HumanLoopConfig (p. 1366) object  
Required: Yes

OutputConfig (p. 885)
An object containing information about where the human review results will be uploaded.
Type: FlowDefinitionOutputConfig (p. 1359) object  
Required: Yes

RoleArn (p. 885)
The Amazon Resource Name (ARN) of the role needed to call other services on your behalf. For example, arn:aws:iam::1234567890:role/service-role/AmazonSageMaker-ExecutionRole-20180111T151298.
Type: String  
Pattern: ^arn:aws[a-zA-Z-]*:iam::\d{12}:role/\?[a-zA-Z0-9+=,.@-_/*]+#$  
Required: Yes

Tags (p. 885)
An array of key-value pairs that contain metadata to help you categorize and organize a flow definition. Each tag consists of a key and a value, both of which you define.
Type: Array of Tag (p. 1517) objects  
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

Response Syntax

{
   "FlowDefinitionArn": "string"
}

Response Elements
If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

FlowDefinitionArn (p. 886)

The Amazon Resource Name (ARN) of the flow definition you create.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:flow-definition/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateHumanTaskUi
Service: Amazon SageMaker Service

Defines the settings you will use for the human review workflow user interface. Reviewers will see a three-panel interface with an instruction area, the item to review, and an input area.

Request Syntax

```
{
  "HumanTaskUiName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "UiTemplate": {
    "Content": "string"
  }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**HumanTaskUiName (p. 888)**

The name of the user interface you are creating.

Type: String


Pattern: `^[a-z0-9](-*[a-z0-9])*$`

Required: Yes

**Tags (p. 888)**

An array of key-value pairs that contain metadata to help you categorize and organize a human review workflow user interface. Each tag consists of a key and a value, both of which you define.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**UiTemplate (p. 888)**

The Liquid template for the worker user interface.

Type: UiTemplate (p. 1569) object

Required: Yes

Response Syntax

```
{
}
```
"HumanTaskUiArn": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**HumanTaskUiArn (p. 888)**

The Amazon Resource Name (ARN) of the human review workflow user interface you create.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:human-task-ui/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateHyperParameterTuningJob

Service: Amazon SageMaker Service

Starts a hyperparameter tuning job. A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm you choose and values for hyperparameters within ranges that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by an objective metric that you choose.

Request Syntax

```json
{
   "HyperParameterTuningJobConfig": {
      "HyperParameterTuningJobObjective": {
         "MetricName": "string",
         "Type": "string"
      },
      "ParameterRanges": {
         "CategoricalParameterRanges": [
            { "Name": "string",
              "Values": [ "string" ]
            }
         ],
         "ContinuousParameterRanges": [
            { "MaxValue": "string",
              "MinValue": "string",
              "Name": "string",
              "ScalingType": "string"
            }
         ],
         "IntegerParameterRanges": [
            { "MaxValue": "string",
              "MinValue": "string",
              "Name": "string",
              "ScalingType": "string"
            }
         ]
      },
      "ResourceLimits": {
         "MaxNumberOfTrainingJobs": number,
         "MaxParallelTrainingJobs": number
      },
      "Strategy": "string",
      "TrainingJobEarlyStoppingType": "string",
      "TuningJobCompletionCriteria": {
         "TargetObjectiveMetricValue": number
      }
   },
   "HyperParameterTuningJobName": "string",
   "Tags": [
      { "Key": "string",
        "Value": "string"
      }
   ],
   "TrainingJobDefinition": {
      "AlgorithmSpecification": {
         "AlgorithmName": "string",
         "MetricDefinitions": [
            { "Name": "string",
              "Type": "string"
            }
         ]
      }
   }
}
```
"Regex": "string"
},
"TrainingImage": "string",
"TrainingInputMode": "string"
},
"CheckpointConfig": {
  "LocalPath": "string",
  "S3Uri": "string"
},
"DefinitionName": "string",
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"HyperParameterRanges": {
  "CategoricalParameterRanges": [
    {
      "Name": "string",
      "Values": [ "string" ]
    }
  ],
  "ContinuousParameterRanges": [
    {
      "MaxValue": "string",
      "MinValue": "string",
      "Name": "string",
      "ScalingType": "string"
    }
  ],
  "IntegerParameterRanges": [
    {
      "MaxValue": "string",
      "MinValue": "string",
      "Name": "string",
      "ScalingType": "string"
    }
  ]
},
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  }
],
"OutputDataConfig": {
  "KmsKeyId": "string"
"S3OutputPath": "string",
},
"ResourceConfig": {
   "InstanceCount": number,
   "InstanceType": "string",
   "VolumeKmsKeyId": "string",
   "VolumeSizeInGB": number
},
"RoleArn": "string",
"StaticHyperParameters": {
   "string": "string"
},
"StoppingCondition": {
   "MaxRuntimeInSeconds": number,
   "MaxWaitTimeInSeconds": number
},
"TuningObjective": {
   "MetricName": "string",
   "Type": "string"
},
"VpcConfig": {
   "SecurityGroupIds": [ "string" ],
   "Subnets": [ "string" ]
}
"TrainingJobDefinitions": [
   {
      "AlgorithmSpecification": {
         "AlgorithmName": "string",
         "MetricDefinitions": [
            {
               "Name": "string",
               "Regex": "string"
            }
         ],
         "TrainingImage": "string",
         "TrainingInputMode": "string"
      },
      "CheckpointConfig": {
         "LocalPath": "string",
         "S3Uri": "string"
      },
      "DefinitionName": "string",
      "EnableInterContainerTrafficEncryption": boolean,
      "EnableManagedSpotTraining": boolean,
      "EnableNetworkIsolation": boolean,
      "HyperParameterRanges": {
         "CategoricalParameterRanges": [
            {
               "Name": "string",
               "Values": [ "string" ]
            }
         ],
         "ContinuousParameterRanges": [
            {
               "MaxValue": "string",
               "MinValue": "string",
               "Name": "string",
               "ScalingType": "string"
            }
         ],
         "IntegerParameterRanges": [
            {
               "MaxValue": "string",
               "MinValue": "string",
               "Name": "string",
            }
         ]
      }
   }
]
"ScalingType": "string"
],
"InputDataConfig": [
{
"ChannelName": "string",
"CompressionType": "string",
"ContentType": "string",
"DataSource": {
"FileSystemDataSource": {
"DirectoryPath": "string",
"FileSystemAccessMode": "string",
"FileSystemId": "string",
"FileSystemType": "string"
},
"S3DataSource": {
"AttributeNames": [ "string" ],
"S3DataDistributionType": "string",
"S3DataType": "string",
"S3Uri": "string"
}
},
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
"Seed": number
}
}
],
"OutputDataConfig": {
"KmsKeyId": "string",
"S3OutputPath": "string"
},
"ResourceConfig": {
"InstanceCount": number,
"InstanceType": "string",
"VolumeKmsKeyId": "string",
"VolumeSizeInGB": number
},
"RoleArn": "string",
"StaticHyperParameters": {
"string": "string"
},
"StoppingCondition": {
"MaxRuntimeInSeconds": number,
"MaxWaitTimeInSeconds": number
},
"TuningObjective": {
"MetricName": "string",
"Type": "string"
},
"VpcConfig": {
"SecurityGroupIds": [ "string" ],
"Subnets": [ "string" ]
}
],
"WarmStartConfig": {
"ParentHyperParameterTuningJobs": [
{
"HyperParameterTuningJobName": "string"
}
],
"WarmStartType": "string"
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

HyperParameterTuningJobConfig (p. 890)

The HyperParameterTuningJobConfig (p. 1389) object that describes the tuning job, including the search strategy, the objective metric used to evaluate training jobs, ranges of parameters to search, and resource limits for the tuning job. For more information, see Perform Automatic Model Tuning (p. 555)

Type: HyperParameterTuningJobConfig (p. 1389) object

Required: Yes

HyperParameterTuningJobName (p. 890)

The name of the tuning job. This name is the prefix for the names of all training jobs that this tuning job launches. The name must be unique within the same AWS account and AWS Region. The name must have {} to {} characters. Valid characters are a-z, A-Z, 0-9, and : + = @ _ % - (hyphen). The name is not case sensitive.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Tags (p. 890)

An array of key-value pairs. You can use tags to categorize your AWS resources in different ways, for example, by purpose, owner, or environment. For more information, see AWS Tagging Strategies.

Tags that you specify for the tuning job are also added to all training jobs that the tuning job launches.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobDefinition (p. 890)

The HyperParameterTrainingJobDefinition (p. 1383) object that describes the training jobs that this tuning job launches, including static hyperparameters, input data configuration, output data configuration, resource configuration, and stopping condition.

Type: HyperParameterTrainingJobDefinition (p. 1383) object

Required: No

TrainingJobDefinitions (p. 890)

Type: Array of HyperParameterTrainingJobDefinition (p. 1383) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.
Required: No  

**WarmStartConfig (p. 890)**

Specifies the configuration for starting the hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric. If you specify `IDENTICAL_DATA_AND_ALGORITHM` as the `WarmStartType` value for the warm start configuration, the training job that performs the best in the new tuning job is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

**Note**

All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.

Type: `HyperParameterTuningJobWarmStartConfig (p. 1394)` object

Required: No

**Response Syntax**

```json
{
   "HyperParameterTuningJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**HyperParameterTuningJobArn (p. 895)**

The Amazon Resource Name (ARN) of the tuning job. Amazon SageMaker assigns an ARN to a hyperparameter tuning job when you create it.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

**Errors**

For information about the errors that are common to all actions, see `Common Errors (p. 1579)`.

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateLabelingJob
Service: Amazon SageMaker Service

Creates a job that uses workers to label the data objects in your input dataset. You can use the labeled data to train machine learning models.

You can select your workforce from one of three providers:

- A private workforce that you create. It can include employees, contractors, and outside experts. Use a private workforce when want the data to stay within your organization or when a specific set of skills is required.
- One or more vendors that you select from the AWS Marketplace. Vendors provide expertise in specific areas.
- The Amazon Mechanical Turk workforce. This is the largest workforce, but it should only be used for public data or data that has been stripped of any personally identifiable information.

You can also use automated data labeling to reduce the number of data objects that need to be labeled by a human. Automated data labeling uses active learning to determine if a data object can be labeled by machine or if it needs to be sent to a human worker. For more information, see Using Automated Data Labeling.

The data objects to be labeled are contained in an Amazon S3 bucket. You create a manifest file that describes the location of each object. For more information, see Using Input and Output Data.

The output can be used as the manifest file for another labeling job or as training data for your machine learning models.

Request Syntax

```
{
  "HumanTaskConfig": {
    "AnnotationConsolidationConfig": {
      "AnnotationConsolidationLambdaArn": "string"
    },
    "MaxConcurrentTaskCount": number,
    "NumberOfHumanWorkersPerDataObject": number,
    "PreHumanTaskLambdaArn": "string",
    "PublicWorkforceTaskPrice": {
      "AmountInUsd": {
        "Cents": number,
        "Dollars": number,
        "TenthFractionsOfACent": number
      }
    },
    "TaskAvailabilityLifetimeInSeconds": number,
    "TaskDescription": "string",
    "TaskKeywords": [ "string" ],
    "TaskTimeLimitInSeconds": number,
    "TaskTitle": "string",
    "UiConfig": {
      "UiTemplateS3Uri": "string"
    },
    "WorkteamArn": "string"
  },
  "InputConfig": {
    "DataAttributes": {
      "ContentClassifiers": [ "string" ]
    },
    "DataSource": {
      "S3DataSource": {
```

897
"ManifestS3Uri": "string"

"LabelAttributeName": "string",
"LabelCategoryConfigS3Uri": "string",
"LabelingJobAlgorithmsConfig": {
  "InitialActiveLearningModelArn": "string",
  "LabelingJobAlgorithmSpecificationArn": "string",
  "LabelingJobResourceConfig": {
    "VolumeKmsKeyId": "string"
  }
},
"LabelingJobName": "string",
"OutputConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"RoleArn": "string",
"StoppingConditions": {
  "MaxHumanLabeledObjectCount": number,
  "MaxPercentageOfInputDatasetLabeled": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
]

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**HumanTaskConfig (p. 897)**

Configures the labeling task and how it is presented to workers; including, but not limited to price, keywords, and batch size (task count).

Type: HumanTaskConfig (p. 1372) object

Required: Yes

**InputConfig (p. 897)**

Input data for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.

Type: LabelingJobInputConfig (p. 1414) object

Required: Yes

**LabelAttributeName (p. 897)**

The attribute name to use for the label in the output manifest file. This is the key for the key/value pair formed with the label that a worker assigns to the object. The name can't end with "-metadata". If you are running a semantic segmentation labeling job, the attribute name must end with "-ref". If you are running any other kind of labeling job, the attribute name must not end with "-ref".

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 

Required: Yes

**LabelCategoryConfigS3Uri (p. 897)**

The S3 URL of the file that defines the categories used to label the data objects.

The file is a JSON structure in the following format:

```json
{
  "document-version": "2018-11-28",
  "labels": [
    {
      "label": "label 1"
    },
    {
      "label": "label 2"
    },
    ...
    {
      "label": "label n"
    }
  ]
}
```

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3):/(/[^/]+)/(.*)$

Required: No

**LabelingJobAlgorithmsConfig (p. 897)**

Configures the information required to perform automated data labeling.

Type: LabelingJobAlgorithmsConfig (p. 1408) object

Required: No

**LabelingJobName (p. 897)**

The name of the labeling job. This name is used to identify the job in a list of labeling jobs.

Type: String

Pattern: ^[a-zA-Z\-0-9](\-[a-zA-Z0-9])*  
Required: Yes

**OutputConfig (p. 897)**

The location of the output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.

Type: LabelingJobOutputConfig (p. 1416) object

Required: Yes

**RoleArn (p. 897)**

The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete data labeling.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9=,\@\-_/]+$  
Required: Yes

**StoppingConditions (p. 897)**

A set of conditions for stopping the labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.

Type: LabelingJobStoppingConditions (p. 1419) object

Required: No

**Tags (p. 897)**

An array of key/value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**Response Syntax**

```
{
  "LabelingJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**LabelingJobArn (p. 900)**

The Amazon Resource Name (ARN) of the labeling job. You use this ARN to identify the labeling job.
Type: String
Length Constraints: Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse
Resource being accessed is in use.
HTTP Status Code: 400

ResourceLimitExceeded
You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateModel
Service: Amazon SageMaker Service

Creates a model in Amazon SageMaker. In the request, you name the model and describe a primary container. For the primary container, you specify the docker image containing inference code, artifacts (from prior training), and custom environment map that the inference code uses when you deploy the model for predictions.

Use this API to create a model if you want to use Amazon SageMaker hosting services or run a batch transform job.

To host your model, you create an endpoint configuration with the CreateEndpointConfig API, and then create an endpoint with the CreateEndpoint API. Amazon SageMaker then deploys all of the containers that you defined for the model in the hosting environment.

To run a batch transform using your model, you start a job with the CreateTransformJob API. Amazon SageMaker uses your model and your dataset to get inferences which are then saved to a specified S3 location.

In the CreateModel request, you must define a container with the PrimaryContainer parameter.

In the request, you also provide an IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute hosting instances or for batch transform jobs. In addition, you also use the IAM role to manage permissions the inference code needs. For example, if the inference code access any other AWS resources, you grant necessary permissions via this role.

Request Syntax

```
{
  "Containers": [
    {
      "ContainerHostname": "string",
      "Environment": {
        "string" : "string"
      },
      "Image": "string",
      "Mode": "string",
      "ModelDataUrl": "string",
      "ModelPackageName": "string"
    }
  ],
  "EnableNetworkIsolation": boolean,
  "ExecutionRoleArn": "string",
  "PrimaryContainer": {
    "ContainerHostname": "string",
    "Environment": {
      "string" : "string"
    },
    "Image": "string",
    "Mode": "string",
    "ModelDataUrl": "string",
    "ModelPackageName": "string"
  },
  "Tags": [ {
    "Key": "string",
    "Value": "string"
  } ],
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ]
  }
}
```
### Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**Containers (p. 902)**

Specifies the containers in the inference pipeline.

- **Type**: Array of ContainerDefinition (p. 1321) objects
- **Array Members**: Maximum number of 5 items.
- **Required**: No

**EnableNetworkIsolation (p. 902)**

Isolates the model container. No inbound or outbound network calls can be made to or from the model container.

- **Type**: Boolean
- **Required**: No

**ExecutionRoleArn (p. 902)**

The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute instances or for batch transform jobs. Deploying on ML compute instances is part of model hosting. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

- **Type**: String
- **Length Constraints**: Minimum length of 20. Maximum length of 2048.
- **Pattern**: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z\-0-9\+=,.@\-_\/]\+$`
- **Required**: Yes

**ModelName (p. 902)**

The name of the new model.

- **Type**: String
- **Length Constraints**: Maximum length of 63.
- **Pattern**: `^[a-zA-Z0-9]+([-\[a-zA-Z0-9]+)*` (not currently active)
- **Required**: Yes

**PrimaryContainer (p. 902)**

The location of the primary docker image containing inference code, associated artifacts, and custom environment map that the inference code uses when the model is deployed for predictions.
Type: ContainerDefinition (p. 1321) object

Required: No

Tags (p. 902)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

VpcConfig (p. 902)

A VpcConfig object that specifies the VPC that you want your model to connect to. Control access to and from your model container by configuring the VPC. VpcConfig is used in hosting services and in batch transform. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

Response Syntax

```
{
  "ModelArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelArn (p. 904)

The ARN of the model created in Amazon SageMaker.

Type: String


Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:model/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateModelPackage
Service: Amazon SageMaker Service

Creates a model package that you can use to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

To create a model package by specifying a Docker container that contains your inference code and the Amazon S3 location of your model artifacts, provide values for `InferenceSpecification`. To create a model from an algorithm resource that you created or subscribed to in AWS Marketplace, provide a value for `SourceAlgorithmSpecification`.

Request Syntax

```json
{
    "CertifyForMarketplace": boolean,
    "InferenceSpecification": {
        "Containers": [
            {
                "ContainerHostname": "string",
                "Image": "string",
                "ImageDigest": "string",
                "ModelDataUrl": "string",
                "ProductId": "string"
            }
        ],
        "SupportedContentTypes": [ "string" ],
        "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
        "SupportedResponseMIMETypes": [ "string" ],
        "SupportedTransformInstanceTypes": [ "string" ]
    },
    "ModelPackageDescription": "string",
    "ModelPackageName": "string",
    "SourceAlgorithmSpecification": {
        "SourceAlgorithms": [
            {
                "AlgorithmName": "string",
                "ModelDataUrl": "string"
            }
        ]
    },
    "ValidationSpecification": {
        "ValidationProfiles": [
            {
                "ProfileName": "string",
                "TransformJobDefinition": {
                    "BatchStrategy": "string",
                    "Environment": {
                        "string": "string"
                    },
                    "MaxConcurrentTransforms": number,
                    "MaxPayloadInMB": number,
                    "TransformInput": {
                        "CompressionType": "string",
                        "ContentType": "string",
                        "DataSource": {
                            "S3DataSource": {
                                "S3DataType": "string",
                                "S3Uri": "string"
                            }
                        },
                        "SplitType": "string"
                    }
                }
            }
        ]
    }
}
```
"TransformOutput": {
    "Accept": "string",
    "AssembleWith": "string",
    "KmsKeyId": "string",
    "S3OutputPath": "string"
},
"TransformResources": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string"
}
}],
"ValidationRole": "string"
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CertifyForMarketplace (p. 906)

Whether to certify the model package for listing on AWS Marketplace.

Type: Boolean

Required: No

InferenceSpecification (p. 906)

Specifies details about inference jobs that can be run with models based on this model package, including the following:

- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the model package supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the model package supports for inference.

Type: InferenceSpecification (p. 1396) object

Required: No

ModelPackageDescription (p. 906)

A description of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

ModelPackageName (p. 906)

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

SourceAlgorithmSpecification (p. 906)

Details about the algorithm that was used to create the model package.

Type: SourceAlgorithmSpecification (p. 1512) object

Required: No

ValidationSpecification (p. 906)

Specifies configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.

Type: ModelPackageValidationSpecification (p. 1434) object

Required: No

Response Syntax

```json
{
   "ModelPackageArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelPackageArn (p. 908)

The Amazon Resource Name (ARN) of the new model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateMonitoringSchedule

Service: Amazon SageMaker Service

Creates a schedule that regularly starts Amazon SageMaker Processing Jobs to monitor the data captured for an Amazon SageMaker Endpoint.

Request Syntax

```
{
  "MonitoringScheduleConfig": {
    "MonitoringJobDefinition": {
      "BaselineConfig": {
        "ConstraintsResource": {
          "S3Uri": "string"
        },
        "StatisticsResource": {
          "S3Uri": "string"
        }
      },
      "Environment": {
        "string": "string"
      },
      "MonitoringAppSpecification": {
        "ContainerArguments": [ "string" ],
        "ContainerEntrypoint": [ "string" ],
        "ImageUri": "string",
        "PostAnalyticsProcessorSourceUri": "string",
        "RecordPreprocessorSourceUri": "string"
      },
      "MonitoringInputs": [ {
        "EndpointInput": {
          "EndpointName": "string",
          "LocalPath": "string",
          "S3DataDistributionType": "string",
          "S3InputMode": "string"
        }
      } ],
      "MonitoringOutputConfig": {
        "KmsKeyId": "string",
        "MonitoringOutputs": [ {
          "S3Output": {
            "LocalPath": "string",
            "S3UploadMode": "string",
            "S3Uri": "string"
          }
        } ]
      },
      "MonitoringResources": {
        "ClusterConfig": {
          "InstanceCount": number,
          "InstanceType": "string",
          "VolumeKmsKeyId": "string",
          "VolumeSizeInGB": number
        }
      },
      "NetworkConfig": {
        "EnableNetworkIsolation": boolean,
        "VpcConfig": {
          "SecurityGroupIds": [ "string" ],
          "Subnets": [ "string" ]
        }
      }
    }
  }
}
```
The request accepts the following data in JSON format.

**MonitoringScheduleConfig (p. 910)**

The configuration object that specifies the monitoring schedule and defines the monitoring job.

Type: MonitoringScheduleConfig (p. 1451) object

Required: Yes

**MonitoringScheduleName (p. 910)**

The name of the monitoring schedule. The name must be unique within an AWS Region within an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*#

Required: Yes

**Tags (p. 910)**

(Optional) An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**Response Syntax**

```json
{
  "RoleArn": "string",
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number
  },
  "ScheduleConfig": {
    "ScheduleExpression": "string"
  },
  "MonitoringScheduleName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```
"MonitoringScheduleArn": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**MonitoringScheduleArn (p. 911)**

The Amazon Resource Name (ARN) of the monitoring schedule.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateNotebookInstance
Service: Amazon SageMaker Service

Creates an Amazon SageMaker notebook instance. A notebook instance is a machine learning (ML) compute instance running on a Jupyter notebook.

In a CreateNotebookInstance request, specify the type of ML compute instance that you want to run. Amazon SageMaker launches the instance, installs common libraries that you can use to explore datasets for model training, and attaches an ML storage volume to the notebook instance.

Amazon SageMaker also provides a set of example notebooks. Each notebook demonstrates how to use Amazon SageMaker with a specific algorithm or with a machine learning framework.

After receiving the request, Amazon SageMaker does the following:

1. Creates a network interface in the Amazon SageMaker VPC.
2. (Optional) If you specified SubnetId, Amazon SageMaker creates a network interface in your own VPC, which is inferred from the subnet ID that you provide in the input. When creating this network interface, Amazon SageMaker attaches the security group that you specified in the request to the network interface that it creates in your VPC.
3. Launches an EC2 instance of the type specified in the request in the Amazon SageMaker VPC. If you specified SubnetId of your VPC, Amazon SageMaker specifies both network interfaces when launching this instance. This enables inbound traffic from your own VPC to the notebook instance, assuming that the security groups allow it.

After creating the notebook instance, Amazon SageMaker returns its Amazon Resource Name (ARN). You can't change the name of a notebook instance after you create it.

After Amazon SageMaker creates the notebook instance, you can connect to the Jupyter server and work in Jupyter notebooks. For example, you can write code to explore a dataset that you can use for model training, train a model, host models by creating Amazon SageMaker endpoints, and validate hosted models.

For more information, see How It Works.

Request Syntax

```json
{
    "AcceleratorTypes": [ "string" ],
    "AdditionalCodeRepositories": [ "string" ],
    "DefaultCodeRepository": "string",
    "DirectInternetAccess": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LifecycleConfigName": "string",
    "NotebookInstanceName": "string",
    "RoleArn": "string",
    "RootAccess": "string",
    "SecurityGroupIds": [ "string" ],
    "SubnetId": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "VolumeSizeInGB": number
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AcceleratorTypes (p. 913)**

A list of Elastic Inference (EI) instance types to associate with this notebook instance. Currently, only one instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge | ml.eia2.medium | ml.eia2.large | ml.eia2.xlarge

Required: No

**AdditionalCodeRepositories (p. 913)**

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://([^/]+)/?(.*)|^[a-zA-Z0-9](-*[a-zA-Z0-9])*$  

Required: No

**DefaultCodeRepository (p. 913)**

A Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String


Pattern: ^https://([^/]+)/?(.*)|^[a-zA-Z0-9](-*[a-zA-Z0-9])*$  

Required: No

**DirectInternetAccess (p. 913)**

Sets whether Amazon SageMaker provides internet access to the notebook instance. If you set this to Disabled this notebook instance will be able to access resources only in your VPC, and will not be able to connect to Amazon SageMaker training and endpoint services unless you configure a NAT Gateway in your VPC.

For more information, see Notebook Instances Are Internet-Enabled by Default. You can set the value of this parameter to Disabled only if you set a value for the SubnetId parameter.
Type: String

Valid Values: Enabled | Disabled

Required: No

**InstanceType (p. 913)**

The type of ML compute instance to launch for the notebook instance.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge

Required: Yes

**KmsKeyId (p. 913)**

The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to your notebook instance. The KMS key you provide must be enabled. For information, see Enabling and Disabling Keys in the AWS Key Management Service Developer Guide.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

**LifecycleConfigName (p. 913)**

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]+$*

Required: No

**NotebookInstanceName (p. 913)**

The name of the new notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]+$*

Required: Yes
**RoleArn (p. 913)**

When you send any requests to AWS resources from the notebook instance, Amazon SageMaker assumes this role to perform tasks on your behalf. You must grant this role necessary permissions so Amazon SageMaker can perform these tasks. The policy must allow the Amazon SageMaker service principal (sagemaker.amazonaws.com) permissions to assume this role. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$`

Required: Yes

**RootAccess (p. 913)**

Whether root access is enabled or disabled for users of the notebook instance. The default value is `Enabled`.

**Note**

Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Type: String

Valid Values: `Enabled` | `Disabled`

Required: No

**SecurityGroupIds (p. 913)**

The VPC security group IDs, in the form sg-xxxxxxxx. The security groups must be for the same VPC as specified in the subnet.

Type: Array of strings

Array Members: Maximum number of 5 items.

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: No

**SubnetId (p. 913)**

The ID of the subnet in a VPC to which you would like to have a connectivity from your ML compute instance.

Type: String

Length Constraints: Maximum length of 32.

Pattern: `[-0-9a-zA-Z]+`

Required: No
**Tags (p. 913)**

A list of tags to associate with the notebook instance. You can add tags later by using the CreateTags API.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**VolumeSizeInGB (p. 913)**

The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB.

Type: Integer


Required: No

**Response Syntax**

```
{
    "NotebookInstanceArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NotebookInstanceArn (p. 917)**

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
CreateNotebookInstanceLifecycleConfig
Service: Amazon SageMaker Service

Creates a lifecycle configuration that you can associate with a notebook instance. A lifecycle configuration is a collection of shell scripts that run when you create or start a notebook instance.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the \$PATH environment variable that is available to both scripts is `/sbin:bin:/usr/sbin:/usr/bin`.


Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```json
{
   "NotebookInstanceLifecycleConfigName": "string",
   "OnCreate": [
      {
         "Content": "string"
      }
   ],
   "OnStart": [
      {
         "Content": "string"
      }
   ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 919)**

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**OnCreate (p. 919)**

A shell script that runs only once, when you create a notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects
Array Members: Maximum number of 1 item.
Required: No

OnStart (p. 919)
A shell script that runs every time you start a notebook instance, including when you create the notebook instance. The shell script must be a base64-encoded string.
Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects
Array Members: Maximum number of 1 item.
Required: No

Response Syntax

```
{
  "NotebookInstanceLifecycleConfigArn": "string"
}
```

Response Elements
If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

NotebookInstanceLifecycleConfigArn (p. 920)
The Amazon Resource Name (ARN) of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 256.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceLimitExceeded
You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
CreatePresignedDomainUrl
Service: Amazon SageMaker Service

Creates a URL for a specified UserProfile in a Domain. When accessed in a web browser, the user will be automatically signed in to Amazon SageMaker Amazon SageMaker Studio (Studio), and granted access to all of the Apps and files associated with that Amazon Elastic File System (EFS). This operation can only be called when AuthMode equals IAM.

Request Syntax

```
{
    "DomainId": "string",
    "SessionExpirationDurationInSeconds": number,
    "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 922)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

**SessionExpirationDurationInSeconds (p. 922)**

The session expiration duration in seconds.

Type: Integer


Required: No

**UserProfileName (p. 922)**

The name of the UserProfile to sign-in as.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```
{
    "AuthorizedUrl": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**AuthorizedUrl (p. 922)**

The presigned URL.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreatePresignedNotebookInstanceUrl
Service: Amazon SageMaker Service

Returns a URL that you can use to connect to the Jupyter server from a notebook instance. In the Amazon SageMaker console, when you choose Open next to a notebook instance, Amazon SageMaker opens a new tab showing the Jupyter server home page from the notebook instance. The console uses this API to get the URL and show the page.

IAM authorization policies for this API are also enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance. For example, you can restrict access to this API and to the URL that it returns to a list of IP addresses that you specify. Use the NotIpAddress condition operator and the aws:SourceIP condition context key to specify the list of IP addresses that you want to have access to the notebook instance. For more information, see Limit Access to a Notebook Instance by IP Address.

Note
The URL that you get from a call to CreatePresignedNotebookInstanceUrl (p. 924) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the AWS console sign-in page.

Request Syntax

```json
{
    "NotebookInstanceName": "string",
    "SessionExpirationDurationInSeconds": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 924)**

The name of the notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]*$ Required: Yes

**SessionExpirationDurationInSeconds (p. 924)**

The duration of the session, in seconds. The default is 12 hours.

Type: Integer


Required: No

Response Syntax

```json
{
}
```
"AuthorizedUrl": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AuthorizedUrl (p. 924)**

A JSON object that contains the URL string.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateProcessingJob
Service: Amazon SageMaker Service

Creates a processing job.

Request Syntax

```json
{
   "AppSpecification": {
      "ContainerArguments": [ "string" ],
      "ContainerEntrypoint": [ "string" ],
      "ImageUri": "string"
   },
   "Environment": {
      "string": "string"
   },
   "ExperimentConfig": {
      "ExperimentName": "string",
      "TrialComponentDisplayName": "string",
      "TrialName": "string"
   },
   "NetworkConfig": {
      "EnableNetworkIsolation": boolean,
      "VpcConfig": {
         "SecurityGroupIds": [ "string" ],
         "Subnets": [ "string" ]
      }
   },
   "ProcessingInputs": [
      {
         "InputName": "string",
         "S3Input": {
            "LocalPath": "string",
            "S3CompressionType": "string",
            "S3DataDistributionType": "string",
            "S3DataType": "string",
            "S3InputMode": "string",
            "S3Uri": "string"
         }
      }
   ],
   "ProcessingJobName": "string",
   "ProcessingOutputConfig": {
      "KmsKeyId": "string",
      "Outputs": [
         {
            "OutputName": "string",
            "S3Output": {
               "LocalPath": "string",
               "S3UploadMode": "string",
               "S3Uri": "string"
            }
         }
      ]
   },
   "ProcessingResources": {
      "ClusterConfig": {
         "InstanceCount": number,
         "InstanceType": "string",
         "VolumeKmsKeyId": "string",
         "VolumeSizeInGB": number
      }
   },
   "RoleArn": "string"
}
```
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AppSpecification (p. 926)

Configures the processing job to run a specified Docker container image.

Type: AppSpecification (p. 1290) object

Required: Yes

Environment (p. 926)

Sets the environment variables in the Docker container.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 256.

Value Pattern: [\S\s]*

Required: No

ExperimentConfig (p. 926)

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

NetworkConfig (p. 926)

Networking options for a processing job.

Type: NetworkConfig (p. 1457) object

Required: No

ProcessingInputs (p. 926)

For each input, data is downloaded from S3 into the processing container before the processing job begins running if "S3InputMode" is set to File.

Type: Array of ProcessingInput (p. 1474) objects
Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

**ProcessingJobName (p. 926)**

The name of the processing job. The name must be unique within an AWS Region in the AWS account.

Type: String


Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$

Required: Yes

**ProcessingOutputConfig (p. 926)**

Output configuration for the processing job.

Type: ProcessingOutputConfig (p. 1478) object

Required: No

**ProcessingResources (p. 926)**

Identifies the resources, ML compute instances, and ML storage volumes to deploy for a processing job. In distributed training, you specify more than one instance.

Type: ProcessingResources (p. 1479) object

Required: Yes

**RoleArn (p. 926)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

Type: String


Pattern: ^arn:aws[a-zA-Z\-]*:iam::\d{12}:role/?[a-zA-Z0-9\-_\=,\.\@\-\_\/%]+$

Required: Yes

**StoppingCondition (p. 926)**

The time limit for how long the processing job is allowed to run.

Type: ProcessingStoppingCondition (p. 1483) object

Required: No

**Tags (p. 926)**

(Optional) An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No
Response Syntax

```json
{
  "ProcessingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ProcessingJobArn (p. 929)**

The Amazon Resource Name (ARN) of the processing job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:processing-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
CreateTrainingJob
Service: Amazon SageMaker Service

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a machine learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.
- **HyperParameters** - Specify these algorithm-specific parameters to enable the estimation of model parameters during training. Hyperparameters can be tuned to optimize this learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see [Algorithms](#).
- **InputDataConfig** - Describes the training dataset and the Amazon S3, EFS, or FSx location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of model training.
- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- **EnableManagedSpotTraining** - Optimize the cost of training machine learning models by up to 80% by using Amazon EC2 Spot instances. For more information, see [Managed Spot Training](#).
- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- **StoppingCondition** - To help cap training costs, use `MaxRuntimeInSeconds` to set a time limit for training. Use `MaxWaitTimeInSeconds` to specify how long you are willing to wait for a managed spot training job to complete.

For more information about Amazon SageMaker, see [How It Works](#).

**Request Syntax**

```json
{
  "AlgorithmSpecification": {
    "AlgorithmName": "string",
    "EnableSageMakerMetricsTimeSeries": boolean,
    "MetricDefinitions": [
      {
        "Name": "string",
        "Regex": "string"
      }
    ],
    "TrainingImage": "string",
    "TrainingInputMode": "string"
  },
  "CheckpointConfig": {
    "LocalPath": "string",
    "S3Uri": "string"
  },
  "DebugHookConfig": {
    "CollectionConfigurations": [
      {
        "CollectionName": "string"
      }
    ]
  }
}
```
"CollectionParameters": {
"string" : "string"
}

"HookParameters": {
"string" : "string"
},
"LocalPath": "string",
"S3OutputPath": "string"
},
"DebugRuleConfigurations": [
{
"InstanceType": "string",
"LocalPath": "string",
"RuleConfigurationName": "string",
"RuleEvaluatorImage": "string",
"RuleParameters": {
"string" : "string"
},
"S3OutputPath": "string",
"VolumeSizeInGB": number
}
],
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"ExperimentConfig": {
"ExperimentName": "string",
"TrialComponentDisplayName": "string",
"TrialName": "string"
},
"HyperParameters": {
"string" : "string"
},
"InputDataConfig": [
{
"ChannelName": "string",
"CompressionType": "string",
"ContentType": "string",
"DataSource": {
"FileSystemDataSource": {
"DirectoryPath": "string",
"FileSystemAccessMode": "string",
"FileSystemId": "string",
"FileSystemType": "string"
},
"S3DataSource": {
"AttributeNames": [ "string" ],
"S3DataDistributionType": "string",
"S3DataType": "string",
"S3Uri": "string"
}
},
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
"Seed": number
}
}
],
"OutputDataConfig": {
"KmsKeyId": "string",
"S3OutputPath": "string"
},
"ResourceConfig": {
"string" : "string"
}


```
"InstanceCount": number,
"InstanceType": "string",
"VolumeKmsKeyId": "string",
"VolumeSizeInGB": number,
"RoleArn": "string",
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
],
"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},
"TrainingJobName": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AlgorithmSpecification (p. 931)**

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 1274) object

Required: Yes

**CheckpointConfig (p. 931)**

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

Required: No

**DebugHookConfig (p. 931)**

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

Required: No

**DebugRuleConfigurations (p. 931)**

Configuration information for debugging rules.
Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

EnableInterContainerTrafficEncryption (p. 931)

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training. For more information, see Protect Communications Between ML Compute Instances in a Distributed Training Job.

Type: Boolean

Required: No

EnableManagedSpotTraining (p. 931)

To train models using managed spot training, choose True. Managed spot training provides a fully managed and scalable infrastructure for training machine learning models. this option is useful when training jobs can be interrupted and when there is flexibility when the training job is run. The complete and intermediate results of jobs are stored in an Amazon S3 bucket, and can be used as a starting point to train models incrementally. Amazon SageMaker provides metrics and logs in CloudWatch. They can be used to see when managed spot training jobs are running, interrupted, resumed, or completed.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 931)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

Required: No

ExperimentConfig (p. 931)

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

HyperParameters (p. 931)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.
InputDataConfig (p. 931)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3, EFS, or FSx location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams. For example, if you specify an EFS location, input data files will be made available as input streams. They do not need to be downloaded.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

OutputDataConfig (p. 931)

Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

ResourceConfig (p. 931)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object

RoleArn (p. 931)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.
Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-\_]/+$

Required: Yes

StoppingCondition (p. 931)
Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object

Required: Yes

Tags (p. 931)
An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TensorBoardOutputConfig (p. 931)
Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

TrainingJobName (p. 931)
The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9-]+(-*[a-zA-Z0-9-]+)*$

Required: Yes

VpcConfig (p. 931)
A VpcConfig (p. 1577) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No
Response Syntax

```json
{
   "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 937)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
CreateTransformJob

Service: Amazon SageMaker Service

Starts a transform job. A transform job uses a trained model to get inferences on a dataset and saves these results to an Amazon S3 location that you specify.

To perform batch transformations, you create a transform job and use the data that you have readily available.

In the request body, you provide the following:

- **TransformJobName** - Identifies the transform job. The name must be unique within an AWS Region in an AWS account.
- **ModelName** - Identifies the model to use. **ModelName** must be the name of an existing Amazon SageMaker model in the same AWS Region and AWS account. For information on creating a model, see [CreateModel](p. 902).
- **TransformInput** - Describes the dataset to be transformed and the Amazon S3 location where it is stored.
- **TransformOutput** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.
- **TransformResources** - Identifies the ML compute instances for the transform job.

For more information about how batch transformation works, see [Batch Transform](#).

**Request Syntax**

```
{
  "BatchStrategy": "string",
  "DataProcessing": {
    "InputFilter": "string",
    "JoinSource": "string",
    "OutputFilter": "string"
  },
  "Environment": {
    "string" : "string"
  },
  "ExperimentConfig": {
    "ExperimentName": "string",
    "TrialComponentDisplayName": "string",
    "TrialName": "string"
  },
  "MaxConcurrentTransforms": number,
  "MaxPayloadInMB": number,
  "ModelName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "TransformInput": {
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "SplitType": "string"
  }
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**BatchStrategy (p. 939)**

Specifies the number of records to include in a mini-batch for an HTTP inference request. A record is a single unit of input data that inference can be made on. For example, a single line in a CSV file is a record.

To enable the batch strategy, you must set the SplitType property of the DataProcessing (p. 1328) object to Line, RecordIO, or TFRecord.

To use only one record when making an HTTP invocation request to a container, set BatchStrategy to SingleRecord and SplitType to Line.

To fit as many records in a mini-batch as can fit within the MaxPayloadInMB limit, set BatchStrategy to MultiRecord and SplitType to Line.

Type: String

Valid Values: MultiRecord | SingleRecord

Required: No

**DataProcessing (p. 939)**

The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.

Type: DataProcessing (p. 1328) object

Required: No

**Environment (p. 939)**

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map
Key Length Constraints: Maximum length of 1024.

Key Pattern: \[a-zA-Z\_][a-zA-Z0-9\_]*

Value Length Constraints: Maximum length of 10240.

Value Pattern: \[\S\s\]*

Required: No

ExperimentConfig (p. 939)

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

MaxConcurrentTransforms (p. 939)

The maximum number of parallel requests that can be sent to each instance in a transform job. If MaxConcurrentTransforms is set to 0 or left unset, Amazon SageMaker checks the optional execution-parameters to determine the settings for your chosen algorithm. If the execution-parameters endpoint is not enabled, the default value is 1. For more information on execution-parameters, see How Containers Serve Requests. For built-in algorithms, you don’t need to set a value for MaxConcurrentTransforms.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MaxPayloadInMB (p. 939)

The maximum allowed size of the payload, in MB. A payload is the data portion of a record (without metadata). The value in MaxPayloadInMB must be greater than, or equal to, the size of a single record. To estimate the size of a record in MB, divide the size of your dataset by the number of records. To ensure that the records fit within the maximum payload size, we recommend using a slightly larger value. The default value is 6 MB.

For cases where the payload might be arbitrarily large and is transmitted using HTTP chunked encoding, set the value to 0. This feature works only in supported algorithms. Currently, Amazon SageMaker built-in algorithms do not support HTTP chunked encoding.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

ModelName (p. 939)

The name of the model that you want to use for the transform job. ModelName must be the name of an existing Amazon SageMaker model within an AWS Region in an AWS account.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z\-\_\-0-9\(-\_\-0-9\)]*(-*[a-zA-Z0-9\-\_\-0-9\)]\)*

Required: Yes
Tags (p. 939)

(Optional) An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TransformInput (p. 939)

Describes the input source and the way the transform job consumes it.

Type: TransformInput (p. 1536) object

Required: Yes

TransformJobName (p. 939)

The name of the transform job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9\-\*][a-zA-Z0-9\-\*]*

Required: Yes

TransformOutput (p. 939)

Describes the results of the transform job.

Type: TransformOutput (p. 1542) object

Required: Yes

TransformResources (p. 939)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: TransformResources (p. 1544) object

Required: Yes

Response Syntax

```
{
  "TransformJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TransformJobArn (p. 942)

The Amazon Resource Name (ARN) of the transform job.
Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being accessed is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateTrial
Service: Amazon SageMaker Service

Creates an Amazon SageMaker trial. A trial is a set of steps called trial components that produce a machine learning model. A trial is part of a single Amazon SageMaker experiment.

When you use Amazon SageMaker Studio or the Amazon SageMaker Python SDK, all experiments, trials, and trial components are automatically tracked, logged, and indexed. When you use the AWS SDK for Python (Boto), you must use the logging APIs provided by the SDK.

You can add tags to a trial and then use the Search (p. 1196) API to search for the tags.

To get a list of all your trials, call the ListTrials (p. 1185) API. To view a trial's properties, call the DescribeTrial (p. 1080) API. To create a trial component, call the CreateTrialComponent (p. 947) API.

Request Syntax

```
{
  "DisplayName": "string",
  "ExperimentName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

DisplayName (p. 944)

The name of the trial as displayed. The name doesn't need to be unique. If DisplayName isn't specified, TrialName is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

ExperimentName (p. 944)

The name of the experiment to associate the trial with.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes
**Tags (p. 944)**

A list of tags to associate with the trial. You can use Search (p. 1196) API to search on the tags.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**TrialName (p. 944)**

The name of the trial. The name must be unique in your AWS account and is not case-sensitive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9\-]*$

Required: Yes

**Response Syntax**

```json
{
   "TrialArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialArn (p. 945)**

The Amazon Resource Name (ARN) of the trial.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:experiment-trial/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**CreateTrialComponent**  
Service: Amazon SageMaker Service

Creates a *trial component*, which is a stage of a machine learning *trial*. A trial is composed of one or more trial components. A trial component can be used in multiple trials.

Trial components include pre-processing jobs, training jobs, and batch transform jobs.

When you use Amazon SageMaker Studio or the Amazon SageMaker Python SDK, all experiments, trials, and trial components are automatically tracked, logged, and indexed. When you use the AWS SDK for Python (Boto), you must use the logging APIs provided by the SDK.

You can add tags to a trial component and then use the [Search (p. 1196)](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Search.html) API to search for the tags.

**Note**  
`CreateTrialComponent` can only be invoked from within an Amazon SageMaker managed environment. This includes Amazon SageMaker training jobs, processing jobs, transform jobs, and Amazon SageMaker notebooks. A call to `CreateTrialComponent` from outside one of these environments results in an error.

**Request Syntax**

```json
{
   "DisplayName": "string",
   "EndTime": number,
   "InputArtifacts": {
      "string": {
         "MediaType": "string",
         "Value": "string"
      }
   },
   "OutputArtifacts": {
      "string": {
         "MediaType": "string",
         "Value": "string"
      }
   },
   "Parameters": {
      "string": {
         "NumberValue": number,
         "StringValue": "string"
      }
   },
   "StartTime": number,
   "Status": {
      "Message": "string",
      "PrimaryStatus": "string"
   },
   "Tags": [ {
      "Key": "string",
      "Value": "string"
   } ],
   "TrialComponentName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters (p. 1581)](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CommonParameters.html).
The request accepts the following data in JSON format.

**DisplayName (p. 947)**

The name of the component as displayed. The name doesn't need to be unique. If `DisplayName` isn't specified, `TrialComponentName` is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*$`

Required: No

**EndTime (p. 947)**

When the component ended.

Type: Timestamp

Required: No

**InputArtifacts (p. 947)**

The input artifacts for the component. Examples of input artifacts are datasets, algorithms, hyperparameters, source code, and instance types.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: `.*`

Required: No

**OutputArtifacts (p. 947)**

The output artifacts for the component. Examples of output artifacts are metrics, snapshots, logs, and images.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: `.*`

Required: No

**Parameters (p. 947)**

The hyperparameters for the component.

Type: String to `TrialComponentParameterValue (p. 1556)` object map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.*`

Required: No

**StartTime (p. 947)**

When the component started.
Type: Timestamp
Required: No

**Status (p. 947)**

The status of the component. States include:
- InProgress
- Completed
- Failed

Type: TrialComponentStatus (p. 1561) object
Required: No

**Tags (p. 947)**

A list of tags to associate with the component. You can use Search (p. 1196) API to search on the tags.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**TrialComponentName (p. 947)**

The name of the component. The name must be unique in your AWS account and is not case-sensitive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9\-_]*[a-zA-Z0-9\-_]*$

Required: Yes

**Response Syntax**

```json
{
    "TrialComponentArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialComponentArn (p. 949)**

The Amazon Resource Name (ARN) of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateUserProfile
Service: Amazon SageMaker Service

Creates a new user profile. A user profile represents a single user within a Domain, and is the main way to reference a "person" for the purposes of sharing, reporting and other user-oriented features. This entity is created during on-boarding. If an administrator invites a person by email or imports them from SSO, a new UserProfile is automatically created. This entity is the primary holder of settings for an individual user and has a reference to the user's private Amazon Elastic File System (EFS) home directory.

Request Syntax

```json
{
    "DomainId": "string",
    "SingleSignOnUserIdentifier": "string",
    "SingleSignOnUserValue": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "UserProfileName": "string",
    "UserSettings": {
        "ExecutionRole": "string",
        "JupyterServerAppSettings": {
            "DefaultResourceSpec": {
                "EnvironmentArn": "string",
                "InstanceType": "string"
            }
        },
        "KernelGatewayAppSettings": {
            "DefaultResourceSpec": {
                "EnvironmentArn": "string",
                "InstanceType": "string"
            }
        },
        "SecurityGroups": [ "string" ],
        "SharingSettings": {
            "NotebookOutputOption": "string",
            "S3KmsKeyId": "string",
            "S3OutputPath": "string"
        },
        "TensorBoardAppSettings": {
            "DefaultResourceSpec": {
                "EnvironmentArn": "string",
                "InstanceType": "string"
            }
        }
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 951)**

The ID of the associated Domain.
Type: String
Length Constraints: Maximum length of 63.
Required: Yes

**SingleSignOnUserIdentifier (p. 951)**

A specifier for the type of value specified in SingleSignOnUserValue. Currently, the only supported value is "UserName". If the Domain's AuthMode is SSO, this field is required. If the Domain's AuthMode is not SSO, this field cannot be specified.

Type: String
Pattern: UserName
Required: No

**SingleSignOnUserValue (p. 951)**

The username of the associated AWS Single Sign-On User for this UserProfile. If the Domain's AuthMode is SSO, this field is required, and must match a valid username of a user in your directory. If the Domain's AuthMode is not SSO, this field cannot be specified.

Type: String
Length Constraints: Maximum length of 256.
Required: No

**Tags (p. 951)**

Each tag consists of a key and an optional value. Tag keys must be unique per resource.

Type: Array of Tag (p. 1517) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**UserProfileName (p. 951)**

A name for the UserProfile.

Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: Yes

**UserSettings (p. 951)**

A collection of settings.

Type: UserSettings (p. 1575) object
Required: No

**Response Syntax**

```json
{
   "UserProfileArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**UserProfileArn (p. 952)**

The user profile Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:user-profile/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateWorkteam

Service: Amazon SageMaker Service

Creates a new work team for labeling your data. A work team is defined by one or more Amazon Cognito user pools. You must first create the user pools before you can create a work team.

You cannot create more than 25 work teams in an account and region.

Request Syntax

```json
{
    "Description": "string",
    "MemberDefinitions": [
        {
            "CognitoMemberDefinition": {
                "ClientId": "string",
                "UserGroup": "string",
                "UserPool": "string"
            }
        }
    ],
    "NotificationConfiguration": {
        "NotificationTopicArn": "string"
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**Description (p. 954)**

A description of the work team.

Type: String


Pattern: .+

Required: Yes

**MemberDefinitions (p. 954)**

A list of MemberDefinition objects that contains objects that identify the Amazon Cognito user pool that makes up the work team. For more information, see Amazon Cognito User Pools.

All of the CognitoMemberDefinition objects that make up the member definition must have the same ClientId and UserPool values.

Type: Array of MemberDefinition (p. 1423) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.
Required: Yes

**NotificationConfiguration (p. 954)**

Configures notification of workers regarding available or expiring work items.

Type: NotificationConfiguration (p. 1463) object

Required: No

**Tags (p. 954)**

An array of key-value pairs.

For more information, see Resource Tag and Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**WorkteamName (p. 954)**

The name of the work team. Use this name to identify the work team.

Type: String


Pattern: ^[a-zA-Z0-9]*(-*[a-zA-Z0-9]*)*

Required: Yes

**Response Syntax**

```
{
    "WorkteamArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**WorkteamArn (p. 955)**

The Amazon Resource Name (ARN) of the work team. You can use this ARN to identify the work team.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:workteam/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteAlgorithm
Service: Amazon SageMaker Service
Removes the specified algorithm from your account.

Request Syntax

```
{
  "AlgorithmName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AlgorithmName (p. 957)**

The name of the algorithm to delete.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteApp
Service: Amazon SageMaker Service

Used to stop and delete an app.

Request Syntax

```json
{
    "AppName": "string",
    "AppType": "string",
    "DomainId": "string",
    "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AppName (p. 958)

The name of the app.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: Yes

AppType (p. 958)

The type of app.
Type: String
Valid Values: JupyterServer | KernelGateway | TensorBoard
Required: Yes

DomainId (p. 958)

The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: Yes

UserProfileName (p. 958)

The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteCodeRepository
Service: Amazon SageMaker Service

Deletes the specified Git repository from your account.

Request Syntax

```json
{
   "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 960)**

The name of the Git repository to delete.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteDomain
Service: Amazon SageMaker Service

Used to delete a domain. If you on-boarded with IAM mode, you will need to delete your domain to on-board again using SSO. Use with caution. All of the members of the domain will lose access to their EFS volume, including data, notebooks, and other artifacts.

Request Syntax

```json
{
    "DomainId": "string",
    "RetentionPolicy": {
        "HomeEfsFileSystem": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 961)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

**RetentionPolicy (p. 961)**

The retention policy for this domain, which specifies which resources will be retained after the Domain is deleted. By default, all resources are retained (not automatically deleted).

Type: RetentionPolicy (p. 1500) object

Required: No

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteEndpoint
Service: Amazon SageMaker Service

Deletes an endpoint. Amazon SageMaker frees up all of the resources that were deployed when the endpoint was created.

Amazon SageMaker retires any custom KMS key grants associated with the endpoint, meaning you don't need to use the RevokeGrant API call.

Request Syntax

```json
{
   "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**EndpointName** (p. 963)

The name of the endpoint that you want to delete.

- Type: String
- Length Constraints: Maximum length of 63.
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
- Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteEndpointConfig
Service: Amazon SageMaker Service

Deletes an endpoint configuration. The DeleteEndpointConfig API deletes only the specified configuration. It does not delete endpoints created using the configuration.

Request Syntax

```json
{
    "EndpointConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

EndpointConfigName (p. 965)

The name of the endpoint configuration that you want to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\*[^a-zA-Z0-9]]*$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteExperiment
Service: Amazon SageMaker Service

Deletes an Amazon SageMaker experiment. All trials associated with the experiment must be deleted first. Use the ListTrials (p. 1185) API to get a list of the trials associated with the experiment.

Request Syntax

```
{
  "ExperimentName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ExperimentName (p. 966)**

- The name of the experiment to delete.
- Type: String
- Length Constraints: Minimum length of 1. Maximum length of 82.
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  
- Required: Yes

Response Syntax

```
{
  "ExperimentArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ExperimentArn (p. 966)**

- The Amazon Resource Name (ARN) of the experiment that is being deleted.
- Type: String
- Length Constraints: Maximum length of 256.
- Pattern: `arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:experiment/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteFlowDefinition
Service: Amazon SageMaker Service

Deletes the specified flow definition.

Request Syntax

```json
{
   "FlowDefinitionName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**FlowDefinitionName (p. 968)**

The name of the flow definition you are deleting.

Type: String


Pattern: `^[a-z0-9](-*[a-z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DeleteModel
Service: Amazon SageMaker Service

Deletes a model. The DeleteModel API deletes only the model entry that was created in Amazon SageMaker when you called the CreateModel API. It does not delete model artifacts, inference code, or the IAM role that you specified when creating the model.

Request Syntax

```
{
   "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

ModelName (p. 970)

The name of the model to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**DeleteModelPackage**
Service: Amazon SageMaker Service

Deletes a model package.

A model package is used to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

**Request Syntax**

```json
{
   "ModelPackageName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ModelPackageName (p. 971)**

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String


Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$

Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteMonitoringSchedule
Service: Amazon SageMaker Service

Deletes a monitoring schedule. Also stops the schedule had not already been stopped. This does not delete the job execution history of the monitoring schedule.

Request Syntax

```
{
  "MonitoringScheduleName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

MonitoringScheduleName (p. 973)

The name of the monitoring schedule to delete.

Type: String


Pattern: `^[a-zA-Z0-9-]*[a-zA-Z0-9]$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DeleteNotebookInstance
Service: Amazon SageMaker Service

Deletes an Amazon SageMaker notebook instance. Before you can delete a notebook instance, you must call the StopNotebookInstance API.

Important
When you delete a notebook instance, you lose all of your data. Amazon SageMaker removes the ML compute instance, and deletes the ML storage volume and the network interface associated with the notebook instance.

Request Syntax

```json
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 975)

  The name of the Amazon SageMaker notebook instance to delete.

  Type: String

  Length Constraints: Maximum length of 63.

  Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

  Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
DeleteNotebookInstanceLifecycleConfig
Service: Amazon SageMaker Service

Deletes a notebook instance lifecycle configuration.

Request Syntax

```
{
    "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 977)**

The name of the lifecycle configuration to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9][-\[a-zA-Z0-9]*[^a-zA-Z0-9]*[^a-zA-Z0-9]*]`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteTags
Service: Amazon SageMaker Service

Deletes the specified tags from an Amazon SageMaker resource.

To list a resource's tags, use the ListTags API.

Note
When you call this API to delete tags from a hyperparameter tuning job, the deleted tags are not removed from training jobs that the hyperparameter tuning job launched before you called this API.

Request Syntax

```json
{
    "ResourceArn": "string",
    "TagKeys": [ "string" ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

ResourceArn (p. 978)

The Amazon Resource Name (ARN) of the resource whose tags you want to delete.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:.*

Required: Yes

TagKeys (p. 978)

An array or one or more tag keys to delete.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 50 items.


Pattern: ^([\p{L}\p{Z}\p{N}_.:/=+\-@]*)$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteTrial
Service: Amazon SageMaker Service

Deletes the specified trial. All trial components that make up the trial must be deleted first. Use the DescribeTrialComponent (p. 1083) API to get the list of trial components.

Request Syntax

```
{
   "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialName (p. 980)**

The name of the trial to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Syntax

```
{
   "TrialArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialArn (p. 980)**

The Amazon Resource Name (ARN) of the trial that is being deleted.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9]*:[0-9]{12}:experiment-trial/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteTrialComponent
Service: Amazon SageMaker Service

Deletes the specified trial component. A trial component must be disassociated from all trials before the trial component can be deleted. To disassociate a trial component from a trial, call the DisassociateTrialComponent (p. 1093) API.

Request Syntax

```json
{
   "TrialComponentName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialComponentName (p. 982)**

- The name of the component to delete.
- Type: String
- Length Constraints: Minimum length of 1. Maximum length of 82.
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  
- Required: Yes

Response Syntax

```json
{
   "TrialComponentArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialComponentArn (p. 982)**

- The Amazon Resource Name (ARN) of the component is being deleted.
- Type: String
- Length Constraints: Maximum length of 256.
- Pattern: `arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*[0-9]{12}:experiment-trial-component/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteUserProfile
Service: Amazon SageMaker Service

Deletes a user profile.

Request Syntax

```json
{
   "DomainId": "string",
   "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

DomainId (p. 984)

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

UserProfileName (p. 984)

The user profile name.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteWorkteam
Service: Amazon SageMaker Service

Deletes an existing work team. This operation can't be undone.

Request Syntax

```json
{
  "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**WorkteamName (p. 986)**

- The name of the work team to delete.
- Type: String
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  
- Required: Yes

Response Syntax

```json
{
  "Success": boolean
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Success (p. 986)**

- Returns `true` if the work team was successfully deleted; otherwise, returns `false`.
- Type: Boolean

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

- You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeAlgorithm

Service: Amazon SageMaker Service

Returns a description of the specified algorithm that is in your account.

Request Syntax

```json
{
  "AlgorithmName":  "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AlgorithmName (p. 988)**

The name of the algorithm to describe.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*)?([a-zA-Z0-9\-]{0,62})(?!-)$

Required: Yes

Response Syntax

```json
{
  "AlgorithmArn":  "string",
  "AlgorithmDescription":  "string",
  "AlgorithmName":  "string",
  "AlgorithmStatus":  "string",
  "AlgorithmStatusDetails": {
    "ImageScanStatuses": [
      {
        "FailureReason":  "string",
        "Name":  "string",
        "Status":  "string"
      }
    ],
    "ValidationStatuses": ["string",
      {
        "FailureReason":  "string",
        "Name":  "string",
        "Status":  "string"
      }
    ]
  },
  "CertifyForMarketplace":  boolean,
  "CreationTime":  number,
  "InferenceSpecification": {
    "Containers": [
      {
        "ContainerHostname":  "string",
```
"Image": "string",
"ImageDigest": "string",
"ModelDataUrl": "string",
"ProductId": "string"
}
",
"SupportedContentTypes": [ "string" ],
"SupportedRealtimeInferenceInstanceTypes": [ "string" ],
"SupportedResponseMIMETypes": [ "string" ],
"SupportedTransformInstanceTypes": [ "string" ]
",
"ProductId": "string",
"TrainingSpecification": {
"MetricDefinitions": [ {
  "Name": "string",
  "Regex": "string"
 }]
",
"SupportedHyperParameters": [ {
  "DefaultValue": "string",
  "Description": "string",
  "IsRequired": boolean,
  "IsTunable": boolean,
  "Name": "string",
  "Range": { 
    "CategoricalParameterRangeSpecification": { 
      "Values": [ "string" ]
    },
    "ContinuousParameterRangeSpecification": { 
      "MaxValue": "string",
      "MinValue": "string"
    },
    "IntegerParameterRangeSpecification": { 
      "MaxValue": "string",
      "MinValue": "string"
    }
  },
  "Type": "string"
 }
",
"SupportedTrainingInstanceTypes": [ "string" ],
"SupportedTuningJobObjectiveMetrics": [ {
  "MetricName": "string",
  "Type": "string"
 }]
",
"SupportsDistributedTraining": boolean,
"TrainingChannels": [ {
  "Description": "string",
  "IsRequired": boolean,
  "Name": "string",
  "SupportedCompressionTypes": [ "string" ],
  "SupportedContentTypes": [ "string" ],
  "SupportedInputModes": [ "string" ]
 }
",
"TrainingImage": "string",
"TrainingImageDigest": "string"
}
",
"ValidationSpecification": { 
  "ValidationProfiles": [ 
  ]
}
"ProfileName": "string",
"TrainingJobDefinition": {
  "HyperParameters": {
    "string": "string"
  },
  "InputChange": {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  },
  "OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number,
    "MaxWaitTimeInSeconds": number
  },
  "TrainingInputMode": "string"
},
"TransformJobDefinition": {
  "BatchStrategy": "string",
  "Environment": {
    "string": "string"
  },
  "MaxConcurrentTransforms": number,
  "MaxPayloadInMB": number,
  "TransformInput": {
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "SplitType": "string"
  },
  "TransformOutput": {
    "string": "string"
  }
}
"Accept": "string",
"AssembleWith": "string",
"KmsKeyId": "string",
"S3OutputPath": "string"
},
"TransformResources": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string"
}
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AlgorithmArn (p. 988)**

The Amazon Resource Name (ARN) of the algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*

**AlgorithmDescription (p. 988)**

A brief summary about the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: \p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

**AlgorithmName (p. 988)**

The name of the algorithm being described.

Type: String


Pattern: ^[a-zA-Z0-9-](-*[a-zA-Z0-9])*$

**AlgorithmStatus (p. 988)**

The current status of the algorithm.

Type: String

Valid Values: Pending | InProgress | Completed | Failed | Deleting

**AlgorithmStatusDetails (p. 988)**

Details about the current status of the algorithm.
Type: AlgorithmStatusDetails (p. 1276) object

CertifyForMarketplace (p. 988)

Whether the algorithm is certified to be listed in AWS Marketplace.

Type: Boolean

CreationTime (p. 988)

A timestamp specifying when the algorithm was created.

Type: Timestamp

InferenceSpecification (p. 988)

Details about inference jobs that the algorithm runs.

Type: InferenceSpecification (p. 1396) object

ProductId (p. 988)

The product identifier of the algorithm.

Type: String

Length Constraints: Maximum length of 256.

Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]$*

TrainingSpecification (p. 988)

Details about training jobs run by this algorithm.

Type: TrainingSpecification (p. 1533) object

ValidationSpecification (p. 988)

Details about configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Type: AlgorithmValidationSpecification (p. 1281) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeApp
Service: Amazon SageMaker Service

Describes the app.

Request Syntax

```
{
  "AppName": "string",
  "AppType": "string",
  "DomainId": "string",
  "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AppName (p. 993)

The name of the app.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: Yes

AppType (p. 993)

The type of app.
Type: String
Valid Values: JupyterServer | KernelGateway | TensorBoard
Required: Yes

DomainId (p. 993)

The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: Yes

UserProfileName (p. 993)

The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
Required: Yes

Response Syntax

```json
{
    "AppArn": "string",
    "AppName": "string",
    "AppType": "string",
    "CreationTime": number,
    "DomainId": "string",
    "FailureReason": "string",
    "LastHealthCheckTimestamp": number,
    "LastUserActivityTimestamp": number,
    "ResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
    },
    "Status": "string",
    "UserProfileName": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AppArn (p. 994)**

The app's Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-zA-Z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:app/.*`

**AppName (p. 994)**

The name of the app.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9]\-*[a-zA-Z0-9]*$`

**AppType (p. 994)**

The type of app.

Type: String

Valid Values: JupyterServer | KernelGateway | TensorBoard

**CreationTime (p. 994)**

The creation time.

Type: Timestamp

**DomainId (p. 994)**

The domain ID.
Type: String
Length Constraints: Maximum length of 63.

**FailureReason (p. 994)**
The failure reason.
Type: String
Length Constraints: Maximum length of 1024.

**LastHealthCheckTimestamp (p. 994)**
The timestamp of the last health check.
Type: Timestamp

**LastUserActivityTimestamp (p. 994)**
The timestamp of the last user's activity.
Type: Timestamp

**ResourceSpec (p. 994)**
The instance type and quantity.
Type: ResourceSpec (p. 1499) object

**Status (p. 994)**
The status.
Type: String
Valid Values: Deleted | Deleting | Failed | InService | Pending

**UserProfileName (p. 994)**
The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9]*([-\[a-zA-Z0-9]*\]]*)

**Errors**
For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**
Resource being access is not found.
HTTP Status Code: 400

**See Also**
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeAutoMLJob

Service: Amazon SageMaker Service

Returns information about an Amazon SageMaker job.

Request Syntax

```json
{
    "AutoMLJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AutoMLJobName (p. 997)**

Request information about a job using that job's unique name.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*$`

Required: Yes

Response Syntax

```json
{
    "AutoMLJobArn": "string",
    "AutoMLJobArtifacts": {
        "CandidateDefinitionNotebookLocation": "string",
        "DataExplorationNotebookLocation": "string"
    },
    "AutoMLJobConfig": {
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": number,
            "MaxCandidates": number,
            "MaxRuntimePerTrainingJobIn Seconds": number
        },
        "SecurityConfig": {
            "EnableInterContainerTrafficEncryption": boolean,
            "VolumeKmsKeyId": "string",
            "VpcConfig": {
                "SecurityGroupId s": [ "string" ],
                "Subnets": [ "string" ]
            }
        }
    },
    "AutoMLJobName": "string",
    "AutoMLJobObjective": {
        "MetricName": "string"
    },
    "AutoMLJobSecondaryStatus": "string",
    "AutoMLJobStatus": "string",
    "BestCandidate": {
        "CandidateName": "string",
        "CandidateStatus": "string",
```
"CandidateSteps": [
  {
    "CandidateStepArn": "string",
    "CandidateStepName": "string",
    "CandidateStepType": "string"
  }
],
"CreationTime": number,
"EndTime": number,
"FailureReason": "string",
"FinalAutoMLJobObjectiveMetric": {
  "MetricName": "string",
  "Type": "string",
  "Value": number
},
"InferenceContainers": [
  {
    "Environment": {
      "string": "string"
    },
    "Image": "string",
    "ModelDataUrl": "string"
  }
],
"LastModifiedTime": number,
"ObjectiveStatus": "string"
},
"CreationTime": number,
"EndTime": number,
"FailureReason": "string",
"GenerateCandidateDefinitionsOnly": boolean,
"InputDataConfig": [
  {
    "CompressionType": "string",
    "DataSource": {
      "S3DataSource": {
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "TargetAttributeName": "string"
  }
],
"LastModifiedTime": number,
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ProblemType": "string",
"ResolvedAttributes": {
  "AutoMLJobObjective": {
    "MetricName": "string"
  },
  "CompletionCriteria": {
    "MaxAutoMLJobRuntimeInSeconds": number,
    "MaxCandidates": number,
    "MaxRuntimePerTrainingJobInSeconds": number
  },
  "ProblemType": "string"
},
"RoleArn": "string"}
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AutoMLJobArn (p. 997)**
- Returns the job's ARN.
  - Type: String
    - Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:automl-job/.*`

**AutoMLJobArtifacts (p. 997)**
- Returns information on the job's artifacts found in AutoMLJobArtifacts.
  - Type: `AutoMLJobArtifacts (p. 1297)` object

**AutoMLJobConfig (p. 997)**
- Returns the job's config.
  - Type: `AutoMLJobConfig (p. 1299)` object

**AutoMLJobName (p. 997)**
- Returns the name of a job.
  - Type: String
    - Pattern: `^[a-zA-Z0-9\-]*[a-zA-Z0-9\-]*$`

**AutoMLJobObjective (p. 997)**
- Returns the job's objective.
  - Type: `AutoMLJobObjective (p. 1300)` object

**AutoMLJobSecondaryStatus (p. 997)**
- Returns the job's AutoMLJobSecondaryStatus.
  - Type: String
  - Valid Values: Starting | AnalyzingData | FeatureEngineering | ModelTuning | MaxCandidatesReached | Failed | Stopped | MaxAutoMLJobRuntimeReached | Stopping | CandidateDefinitionsGenerated

**AutoMLJobStatus (p. 997)**
- Returns the job's AutoMLJobStatus.
  - Type: String
  - Valid Values: Completed | InProgress | Failed | Stopped | Stopping

**BestCandidate (p. 997)**
- Returns the job's BestCandidate.
  - Type: `AutoMLCandidate (p. 1291)` object
CreationTime (p. 997)
Returns the job's creation time.
Type: Timestamp

EndTime (p. 997)
Returns the job's end time.
Type: Timestamp

FailureReason (p. 997)
Returns the job's FailureReason.
Type: String
Length Constraints: Maximum length of 1024.

GenerateCandidateDefinitionsOnly (p. 997)
Returns the job's output from GenerateCandidateDefinitionsOnly.
Type: Boolean

InputDataConfig (p. 997)
Returns the job's input data config.
Type: Array of AutoMLChannel (p. 1294) objects

InputDataConfig (p. 997)
Type: Array of AutoMLInputDataConfig (p. 1303) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

LastModifiedTime (p. 997)
Returns the job's last modified time.
Type: Timestamp

OutputDataConfig (p. 997)
Returns the job's output data config.
Type: AutoMLOutputDataConfig (p. 1303) object

ProblemType (p. 997)
Returns the job's problem type.
Type: String
Valid Values: BinaryClassification | MulticlassClassification | Regression

ResolvedAttributes (p. 997)
This contains ProblemType, AutoMLJobObjective and CompletionCriteria. They're auto-inferred values, if not provided by you. If you do provide them, then they'll be the same as provided.
Type: ResolvedAttributes (p. 1495) object

RoleArn (p. 997)
The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that has read permission to the input data location and write permission to the output data location in Amazon S3.
Type: String

Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\//]+$  

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNot Found

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeCodeRepository
Service: Amazon SageMaker Service

Gets details about the specified Git repository.

Request Syntax

```json
{
    "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 1002)

    The name of the Git repository to describe.

    Type: String


    Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

    Required: Yes

Response Syntax

```json
{
    "CodeRepositoryArn": "string",
    "CodeRepositoryName": "string",
    "CreationTime": number,
    "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
    },
    "LastModifiedTime": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CodeRepositoryArn (p. 1002)

    The Amazon Resource Name (ARN) of the Git repository.

    Type: String

    Length Constraints: Minimum length of 1. Maximum length of 2048.

    Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*
CodeRepositoryName (p. 1002)
The name of the Git repository.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

CreationTime (p. 1002)
The date and time that the repository was created.
Type: Timestamp

GitConfig (p. 1002)
Configuration details about the repository, including the URL where the repository is located, the
default branch, and the Amazon Resource Name (ARN) of the AWS Secrets Manager secret that
contains the credentials used to access the repository.
Type: GitConfig (p. 1362) object

LastModifiedTime (p. 1002)
The date and time that the repository was last changed.
Type: Timestamp

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**DescribeCompilationJob**

Service: Amazon SageMaker Service

Returns information about a model compilation job.

To create a model compilation job, use `CreateCompilationJob` (p. 868). To get information about multiple model compilation jobs, use `ListCompilationJobs` (p. 1112).

**Request Syntax**

```json
{
   "CompilationJobName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CompilationJobName (p. 1004)**

The name of the model compilation job that you want information about.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**Response Syntax**

```json
{
   "CompilationEndTime": number,
   "CompilationJobArn": "string",
   "CompilationJobName": "string",
   "CompilationJobStatus": "string",
   "CompilationStartTime": number,
   "CreationTime": number,
   "FailureReason": "string",
   "InputConfig": {
      "DataInputConfig": "string",
      "Framework": "string",
      "S3Uri": "string"
   },
   "LastModifiedTime": number,
   "ModelArtifacts": {
      "S3ModelArtifacts": "string"
   },
   "OutputConfig": {
      "S3OutputLocation": "string",
      "TargetDevice": "string"
   },
   "RoleArn": "string",
   "StoppingCondition": {
      "MaxRuntimeInSeconds": number,
      "MaxWaitTimeInSeconds": number
   }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CompilationEndTime (p. 1004)**

The time when the model compilation job on a compilation job instance ended. For a successful or stopped job, this is when the job's model artifacts have finished uploading. For a failed job, this is when Amazon SageMaker detected that the job failed.

Type: Timestamp

**CompilationJobArn (p. 1004)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker assumes to perform the model compilation job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

**CompilationJobName (p. 1004)**

The name of the model compilation job.

Type: String


Pattern: `^[a-zA-Z0-9\-]*\[a-zA-Z0-9\-]*$`

**CompilationJobStatus (p. 1004)**

The status of the model compilation job.

Type: String

Valid Values: INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED

**CompilationStartTime (p. 1004)**

The time when the model compilation job started the CompilationJob instances.

You are billed for the time between this timestamp and the timestamp in the `DescribeCompilationJob:CompilationEndTime (p. 1005)` field. In Amazon CloudWatch Logs, the start time might be later than this time. That's because it takes time to download the compilation job, which depends on the size of the compilation job container.

Type: Timestamp

**CreationTime (p. 1004)**

The time that the model compilation job was created.

Type: Timestamp

**FailureReason (p. 1004)**

If a model compilation job failed, the reason it failed.
Type: String

Length Constraints: Maximum length of 1024.

**InputConfig (p. 1004)**

Information about the location in Amazon S3 of the input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: InputConfig (p. 1398) object

**LastModifiedTime (p. 1004)**

The time that the status of the model compilation job was last modified.

Type: Timestamp

**ModelArtifacts (p. 1004)**

Information about the location in Amazon S3 that has been configured for storing the model artifacts used in the compilation job.

Type: ModelArtifacts (p. 1426) object

**OutputConfig (p. 1004)**

Information about the output location for the compiled model and the target device that the model runs on.

Type: OutputConfig (p. 1465) object

**RoleArn (p. 1004)**

The Amazon Resource Name (ARN) of the model compilation job.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$

**StoppingCondition (p. 1004)**

Specifies a limit to how long a model compilation job can run. When the job reaches the time limit, Amazon SageMaker ends the compilation job. Use this API to cap model training costs.

Type: StoppingCondition (p. 1513) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeDomain
Service: Amazon SageMaker Service

The description of the domain.

Request Syntax

```
{
  "DomainId": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 1008)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

Response Syntax

```
{
  "AuthMode": "string",
  "CreationTime": number,
  "DefaultUserSettings": {
    "ExecutionRole": "string",
    "JupyterServerAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "KernelGatewayAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "SecurityGroups": [ "string" ],
    "SharingSettings": {
      "NotebookOutputOption": "string",
      "S3KeyId": "string",
      "S3OutputPath": "string"
    },
    "TensorBoardAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    }
  }
}
```
"DomainArn": "string",
"DomainId": "string",
"DomainName": "string",
"FailureReason": "string",
"HomeEfsFileSystemId": "string",
"HomeEfsFileSystemKmsKeyId": "string",
"LastModifiedTime": number,
"SingleSignOnManagedApplicationInstanceId": "string",
"Status": "string",
"SubnetIds": [ "string" ],
"Url": "string",
"VpcId": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AuthMode (p. 1008)**

The domain's authentication mode.

Type: String

Valid Values: SSO | IAM

**CreationTime (p. 1008)**

The creation time.

Type: Timestamp

**DefaultUserSettings (p. 1008)**

Settings which are applied to all UserProfile in this domain, if settings are not explicitly specified in a given UserProfile.

Type: UserSettings (p. 1575) object

**DomainArn (p. 1008)**

The domain's Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:domain/.*

**DomainId (p. 1008)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

**DomainName (p. 1008)**

The domain name.

Type: String

Length Constraints: Maximum length of 63.
FailureReason (p. 1008)
The failure reason.
Type: String
Length Constraints: Maximum length of 1024.

HomeEfsFileSystemId (p. 1008)
The ID of the Amazon Elastic File System (EFS) managed by this Domain.
Type: String
Length Constraints: Maximum length of 32.

HomeEfsFileSystemKmsKeyId (p. 1008)
The AWS Key Management Service encryption key ID.
Type: String
Length Constraints: Maximum length of 2048.

LastModifiedTime (p. 1008)
The last modified time.
Type: Timestamp

SingleSignOnManagedApplicationInstanceId (p. 1008)
The SSO managed application instance ID.
Type: String
Length Constraints: Maximum length of 256.

Status (p. 1008)
The status.
Type: String
Valid Values: Deleting | Failed | InService | Pending

SubnetIds (p. 1008)
Security setting to limit to a set of subnets.
Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 16 items.
Length Constraints: Maximum length of 32.

Url (p. 1008)
The domain's URL.
Type: String
Length Constraints: Maximum length of 1024.

**VpcId (p. 1008)**

The ID of the Amazon Virtual Private Cloud.

Type: String

Length Constraints: Maximum length of 32.

Pattern: \([-0-9a-zA-Z]+\]

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeEndpoint
Service: Amazon SageMaker Service

Returns the description of an endpoint.

Request Syntax

```
{
  "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**EndpointName (p. 1012)**

The name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Syntax

```
{
  "CreationTime": number,
  "DataCaptureConfig": {
    "CaptureStatus": "string",
    "CurrentSamplingPercentage": number,
    "DestinationS3Uri": "string",
    "EnableCapture": boolean,
    "KmsKeyId": "string"
  },
  "EndpointArn": "string",
  "EndpointConfigName": "string",
  "EndpointName": "string",
  "EndpointStatus": "string",
  "FailureReason": "string",
  "LastModifiedTime": number,
  "ProductionVariants": [
    {
      "CurrentInstanceCount": number,
      "CurrentWeight": number,
      "DeployedImages": [
        {
          "ResolutionTime": number,
          "ResolvedImage": "string",
          "SpecifiedImage": "string"
        }
      ],
      "DesiredInstanceCount": number,
      "DesiredWeight": number,
    }
  ]
}```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

CreationTime (p. 1012)

A timestamp that shows when the endpoint was created.
Type: Timestamp

DataCaptureConfig (p. 1012)

Type: DataCaptureConfigSummary (p. 1327) object

EndpointArn (p. 1012)

The Amazon Resource Name (ARN) of the endpoint.
Type: String
Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:endpoint/.*

EndpointConfigName (p. 1012)

The name of the endpoint configuration associated with this endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9\-\(\)\[\]\*\+\-\_0-9]*

EndpointName (p. 1012)

Name of the endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9\-\(\)\[\]\*\+\-\_0-9]*

EndpointStatus (p. 1012)

The status of the endpoint.
• OutOfService: Endpoint is not available to take incoming requests.
• Creating: CreateEndpoint (p. 875) is executing.
• Updating: UpdateEndpoint (p. 1233) or UpdateEndpointWeightsAndCapacities (p. 1235) is executing.
• SystemUpdating: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
• RollingBack: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes,
InService: Endpoint is available to process incoming requests.

Deleting: DeleteEndpoint (p. 963) is executing.

Failed: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 1014) for information about the failure. DeleteEndpoint (p. 963) is the only operation that can be performed on a failed endpoint.

FailureReason (p. 1012)

If the status of the endpoint is Failed, the reason why it failed.

Type: String

Valid Values: OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed

LastModifiedTime (p. 1012)

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

ProductionVariants (p. 1012)

An array of ProductionVariantSummary (p. 1486) objects, one for each model hosted behind this endpoint.

Type: Array of ProductionVariantSummary (p. 1486) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeEndpointConfig
Service: Amazon SageMaker Service

Returns the description of an endpoint configuration created using the CreateEndpointConfig API.

Request Syntax

```
{
    "EndpointConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 1015)**

The name of the endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Syntax

```
{
    "CreationTime": number,
    "DataCaptureConfig": {
        "CaptureContentTypeHeader": {
            "CsvContentTypes": [ "string" ],
            "JsonContentTypes": [ "string" ]
        },
        "CaptureOptions": [
            { "CaptureMode": "string"
            }
        ],
        "DestinationS3Uri": "string",
        "EnableCapture": boolean,
        "InitialSamplingPercentage": number,
        "KmsKeyId": "string"
    },
    "EndpointConfigArn": "string",
    "EndpointConfigName": "string",
    "KmsKeyId": "string",
    "ProductionVariants": [
        { "AcceleratorType": "string",
          "InitialInstanceCount": number,
          "InitialVariantWeight": number,
          "InstanceType": "string",
          "ModelName": "string",
          "VariantName": "string"
        }
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CreationTime (p. 1015)**

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

**DataCaptureConfig (p. 1015)**

Type: `DataCaptureConfig` (p. 1326) object

**EndpointConfigArn (p. 1015)**

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*`

**EndpointConfigName (p. 1015)**

Name of the Amazon SageMaker endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9\-]*[a-zA-Z0-9\-]*$`

**KmsKeyId (p. 1015)**

AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.\`

**ProductionVariants (p. 1015)**

An array of `ProductionVariant` objects, one for each model that you want to host at this endpoint.

Type: Array of `ProductionVariant` (p. 1484) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeExperiment
Service: Amazon SageMaker Service

Provides a list of an experiment's properties.

Request Syntax

```json
{
  "ExperimentName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ExperimentName (p. 1018)**

The name of the experiment to describe.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)`

Required: Yes

Response Syntax

```json
{
  "CreatedBy": {
    "DomainId": "string",
    "UserProfileArn": "string",
    "UserProfileName": "string"
  },
  "CreationTime": number,
  "Description": "string",
  "DisplayName": "string",
  "ExperimentArn": "string",
  "ExperimentName": "string",
  "LastModifiedBy": {
    "DomainId": "string",
    "UserProfileArn": "string",
    "UserProfileName": "string"
  },
  "LastModifiedTime": number,
  "Source": {
    "SourceArn": "string",
    "SourceType": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CreatedBy (p. 1018)**

Who created the experiment.

Type: UserContext (p. 1572) object

**CreationTime (p. 1018)**

When the experiment was created.

Type: Timestamp

**Description (p. 1018)**

The description of the experiment.

Type: String

Length Constraints: Maximum length of 3072.

Pattern: .*

**DisplayName (p. 1018)**

The name of the experiment as displayed. If `DisplayName` isn't specified, `ExperimentName` is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9]-*[a-zA-Z0-9]*

**ExperimentArn (p. 1018)**

The Amazon Resource Name (ARN) of the experiment.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:experiment/.*

**ExperimentName (p. 1018)**

The name of the experiment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9\-]+*[a-zA-Z0-9\-]*

**LastModifiedBy (p. 1018)**

Who last modified the experiment.

Type: UserContext (p. 1572) object

**LastModifiedTime (p. 1018)**

When the experiment was last modified.

Type: Timestamp
Source (p. 1018)

The ARN of the source and, optionally, the type.

Type: ExperimentSource (p. 1349) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeFlowDefinition
Service: Amazon SageMaker Service

Returns information about the specified flow definition.

Request Syntax

```json
{
  "FlowDefinitionName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**FlowDefinitionName (p. 1021)**

The name of the flow definition.

Type: String


Pattern: ^[a-z0-9]-*[a-z0-9]*

Required: Yes

Response Syntax

```json
{
  "CreationTime": number,
  "FailureReason": "string",
  "FlowDefinitionArn": "string",
  "FlowDefinitionName": "string",
  "FlowDefinitionStatus": "string",
  "HumanLoopActivationConfig": {
    "HumanLoopActivationConditionsConfig": {
      "HumanLoopActivationConditions": "string"
    },
    "HumanLoopRequestSource": {
      "AwsManagedHumanLoopRequestSource": "string"
    }
  },
  "HumanLoopConfig": {
    "HumanTaskUiArn": "string",
    "PublicWorkforceTaskPrice": {
      "AmountInUsd": {
        "Cents": number,
        "Dollars": number,
        "TenthFractionsOfACent": number
      }
    },
    "TaskAvailabilityLifetimeInSeconds": number,
    "TaskCount": number,
    "TaskDescription": "string",
    "TaskKeywords": [ "string" ],
    "TaskTimeLimitInSeconds": number,
  }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CreationTime (p. 1021)**

The timestamp when the flow definition was created.

Type: Timestamp

**FailureReason (p. 1021)**

Type: String

Length Constraints: Maximum length of 1024.

**FlowDefinitionArn (p. 1021)**

The Amazon Resource Name (ARN) of the flow definition.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:flow-definition/.*`

**FlowDefinitionName (p. 1021)**

The Amazon Resource Name (ARN) of the flow definition.

Type: String


Pattern: `^[a-z0-9](-*[a-z0-9])*

**FlowDefinitionStatus (p. 1021)**

The status of the flow definition. Valid values are listed below.

Type: String

Valid Values: Initializing | Active | Failed | Deleting | Deleted

**HumanLoopActivationConfig (p. 1021)**

An object containing information about what triggers a human review workflow.

Type: `HumanLoopActivationConfig (p. 1365)` object

**HumanLoopConfig (p. 1021)**

An object containing information about who works on the task, the workforce task price, and other task details.
Type: HumanLoopConfig (p. 1366) object

OutputConfig (p. 1021)

An object containing information about the output file.

Type: FlowDefinitionOutputConfig (p. 1359) object

RoleArn (p. 1021)

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) execution role for the flow definition.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\/]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeHumanTaskUi

Service: Amazon SageMaker Service

Returns information about the requested human task user interface.

Request Syntax

```json
{
  "HumanTaskUiName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**HumanTaskUiName (p. 1024)**

The name of the human task user interface you want information about.

Type: String


Pattern: `^[a-z0-9]-*[a-z0-9]*`

Required: Yes

Response Syntax

```json
{
  "CreationTime": number,
  "HumanTaskUiArn": "string",
  "HumanTaskUiName": "string",
  "UiTemplate": {
    "ContentSha256": "string",
    "Url": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CreationTime (p. 1024)**

The timestamp when the human task user interface was created.

Type: Timestamp

**HumanTaskUiArn (p. 1024)**

The Amazon Resource Name (ARN) of the human task user interface.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:human-task-ui/.*

**HumanTaskUiName (p. 1024)**

The name of the human task user interface.

Type: String


Pattern:^[a-z0-9](-*[a-z0-9])*  

**UiTemplate (p. 1024)**

Container for user interface template information.

Type: [UiTemplateInfo (p. 1570)] object

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 1579)].

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeHyperParameterTuningJob
Service: Amazon SageMaker Service

Gets a description of a hyperparameter tuning job.

Request Syntax

```json
{
  "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**HyperParameterTuningJobName (p. 1026)**

The name of the tuning job to describe.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Syntax

```json
{
  "BestTrainingJob": {
    "CreationTime": number,
    "FailureReason": "string",
    "FinalHyperParameterTuningJobObjectiveMetric": {
      "MetricName": "string",
      "Type": "string",
      "Value": number
    },
    "ObjectiveStatus": "string",
    "TrainingEndTime": number,
    "TrainingJobArn": "string",
    "TrainingJobDefinitionName": "string",
    "TrainingJobName": "string",
    "TrainingJobStatus": "string",
    "TrainingStartTime": number,
    "TunedHyperParameters": {
      "string": "string"
    },
    "TuningJobName": "string"
  },
  "CreationTime": number,
  "FailureReason": "string",
  "HyperParameterTuningEndTime": number,
  "HyperParameterTuningJobArn": "string",
  "HyperParameterTuningJobConfig": {
    "HyperParameterTuningJobObjective": {
```

1026
"MetricName": "string",
"Type": "string"
},
"ParameterRanges": {
"CategoricalParameterRanges": [
{
"Name": "string",
"Values": [ "string" ]
}
],
"ContinuousParameterRanges": [
{
"MaxValue": "string",
"MinValue": "string",
"Name": "string",
"ScalingType": "string"
}
],
"IntegerParameterRanges": [
{
"MaxValue": "string",
"MinValue": "string",
"Name": "string",
"ScalingType": "string"
}
]
},
"ResourceLimits": {
"MaxNumberOfTrainingJobs": number,
"MaxParallelTrainingJobs": number
},
"Strategy": "string",
"TrainingJobEarlyStoppingType": "string",
"TuningJobCompletionCriteria": {
"TargetObjectiveMetricValue": number
}
},
"HyperParameterTuningJobName": "string",
"HyperParameterTuningJobStatus": "string",
"LastModifiedTime": number,
"ObjectiveStatusCounters": {
"Failed": number,
"Pending": number,
"Succeeded": number
}
},
"OverallBestTrainingJob": {
"CreationTime": number,
"FailureReason": "string",
"FinalHyperParameterTuningJobObjectiveMetric": {
"MetricName": "string",
"Type": "string",
"Value": number
},
"ObjectiveStatus": "string",
"TrainingEndTime": number,
"TrainingJobArn": "string",
"TrainingJobDefinitionName": "string",
"TrainingJobName": "string",
"TrainingJobStatus": "string",
"TrainingStartTime": number,
"TunedHyperParameters": {
"string": "string"
},
"TuningJobName": "string"
},
"TrainingJobDefinition": {
"MetricName": "string",
"Type": "string"
},
"TrainingJobDefinition": {
"MetricName": "string",
"Type": "string"
}
"AlgorithmSpecification": {
  "AlgorithmName": "string",
  "MetricDefinitions": [
    {
      "Name": "string",
      "Regex": "string"
    }
  ],
  "TrainingImage": "string",
  "TrainingInputMode": "string"
},
"CheckpointConfig": {
  "LocalPath": "string",
  "S3Uri": "string"
},
"DefinitionName": "string",
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"HyperParameterRanges": {
  "CategoricalParameterRanges": [
    {
      "Name": "string",
      "Values": [ "string" ]
    }
  ],
  "ContinuousParameterRanges": [
    {
      "MaxValue": "string",
      "MinValue": "string",
      "Name": "string",
      "ScalingType": "string"
    }
  ],
  "IntegerParameterRanges": [
    {
      "MaxValue": "string",
      "MinValue": "string",
      "Name": "string",
      "ScalingType": "string"
    }
  ]
},
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  }
}


```
{
  "OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "RoleArn": "string",
  "StaticHyperParameters": {
    "string": "string"
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number,
    "MaxWaitTimeInSeconds": number
  },
  "TuningObjective": {
    "MetricName": "string",
    "Type": "string"
  },
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "Subnets": [ "string" ]
  },
  "TrainingJobDefinitions": [
    {
      "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
          {
            "Name": "string",
            "Regex": "string"
          }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
      },
      "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
      },
      "DefinitionName": "string",
      "EnableInterContainerTrafficEncryption": boolean,
      "EnableManagedSpotTraining": boolean,
      "EnableNetworkIsolation": boolean,
      "HyperParameterRanges": {
        "CategoricalParameterRanges": [
          {
            "Name": "string",
            "Values": [ "string" ]
          }
        ],
        "ContinuousParameterRanges": [
          {
            "MaxValue": "string",
            "MinValue": "string",
            "Name": "string",
            "ScalingType": "string"
          }
        ]
      }
    }
  ]
}
```
"IntegerParameterRanges": [
  {
    "MaxValue": "string",
    "MinValue": "string",
    "Name": "string",
    "ScalingType": "string"
  }
],
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": ["string"],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  }
],
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"StaticHyperParameters": {
  "string": "string"
},
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"TuningObjective": {
  "MetricName": "string",
  "Type": "string"
},
"VpcConfig": {
  "SecurityGroupIds": ["string"],
  "Subnets": ["string"
}
],
"TrainingJobStatusCounters": {
  "Completed": number,
  "InProgress": number,
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**BestTrainingJob (p. 1026)**

A TrainingJobSummary (p. 1531) object that describes the training job that completed with the best current HyperParameterTuningJobObjective (p. 1391).

Type: HyperParameterTrainingJobSummary (p. 1386) object

**CreationTime (p. 1026)**

The date and time that the tuning job started.

Type: Timestamp

**FailureReason (p. 1026)**

If the tuning job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**HyperParameterTuningEndTime (p. 1026)**

The date and time that the tuning job ended.

Type: Timestamp

**HyperParameterTuningJobArn (p. 1026)**

The Amazon Resource Name (ARN) of the tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

**HyperParameterTuningJobConfig (p. 1026)**

The HyperParameterTuningJobConfig (p. 1389) object that specifies the configuration of the tuning job.

Type: HyperParameterTuningJobConfig (p. 1389) object
HyperParameterTuningJobName (p. 1026)

The name of the tuning job.
Type: String
Pattern: ^[a-zA-Z0-9-]*(?!-[a-zA-Z0-9-]*))*

HyperParameterTuningJobStatus (p. 1026)

The status of the tuning job: InProgress, Completed, Failed, Stopping, or Stopped.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping

LastModifiedTime (p. 1026)

The date and time that the status of the tuning job was modified.
Type: Timestamp

ObjectiveStatusCounters (p. 1026)

The ObjectiveStatusCounters (p. 1464) object that specifies the number of training jobs, categorized by the status of their final objective metric, that this tuning job launched.
Type: ObjectiveStatusCounters (p. 1464) object

OverallBestTrainingJob (p. 1026)

If the hyperparameter tuning job is an warm start tuning job with a WarmStartType of IDENTICAL_DATA_AND_ALGORITHM, this is the TrainingJobSummary (p. 1531) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.
Type: HyperParameterTrainingJobSummary (p. 1386) object

TrainingJobDefinition (p. 1026)

The HyperParameterTrainingJobDefinition (p. 1383) object that specifies the definition of the training jobs that this tuning job launches.
Type: HyperParameterTrainingJobDefinition (p. 1383) object

TrainingJobDefinitions (p. 1026)

Type: Array of HyperParameterTrainingJobDefinition (p. 1383) objects
Array Members: Minimum number of 1 item. Maximum number of 10 items.

TrainingJobStatusCounters (p. 1026)

The TrainingJobStatusCounters (p. 1529) object that specifies the number of training jobs, categorized by status, that this tuning job launched.
Type: TrainingJobStatusCounters (p. 1529) object

WarmStartConfig (p. 1026)

The configuration for starting the hyperparameter parameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.
Type: HyperParameterTuningJobWarmStartConfig (p. 1394) object
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeLabelingJob
Service: Amazon SageMaker Service

Gets information about a labeling job.

Request Syntax

```
{
   "LabelingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**LabelingJobName (p. 1034)**

The name of the labeling job to return information for.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Syntax

```
{
   "CreationTime": number,
   "FailureReason": "string",
   "HumanTaskConfig": {
      "AnnotationConsolidationConfig": {
         "AnnotationConsolidationLambdaArn": "string"
      },
      "MaxConcurrentTaskCount": number,
      "NumberOfHumanWorkersPerDataObject": number,
      "PreHumanTaskLambdaArn": "string",
      "PublicWorkforceTaskPrice": {
         "AmountInUsd": {
            "Cents": number,
            "Dollars": number,
            "TenthFractionsOfACent": number
         }
      },
      "TaskAvailabilityLifetimeInSeconds": number,
      "TaskDescription": "string",
      "TaskKeywords": [ "string" ],
      "TaskTimeLimitInSeconds": number,
      "TaskTitle": "string",
      "UiConfig": {
         "UiTemplateS3Uri": "string"
      },
      "WorkteamArn": "string"
   },
   "InputConfig": {
      "DataAttributes": {
```
"ContentClassifiers": [ "string" ]
},
"DataSource": {
  "S3DataSource": {
    "ManifestS3Uri": "string"
  }
}
},
"JobReferenceCode": "string",
"LabelAttributeName": "string",
"LabelCategoryConfigS3Uri": "string",
"LabelCounters": {
  "FailedNonRetryableError": number,
  "HumanLabeled": number,
  "MachineLabeled": number,
  "TotalLabeled": number,
  "Unlabeled": number
},
"LabelingJobAlgorithmsConfig": {
  "InitialActiveLearningModelArn": "string",
  "LabelingJobAlgorithmSpecificationArn": "string",
  "LabelingJobResourceConfig": {
    "VolumeKmsKeyId": "string"
  }
},
"LabelingJobArn": "string",
"LabelingJobName": "string",
"LabelingJobOutput": {
  "FinalActiveLearningModelArn": "string",
  "OutputDatasetS3Uri": "string"
},
"LabelingJobStatus": "string",
"LastModifiedTime": number,
"OutputConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"RoleArn": "string",
"StoppingConditions": {
  "MaxHumanLabeledObjectCount": number,
  "MaxPercentageOfInputDatasetLabeled": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 1034)

The date and time that the labeling job was created.

Type: Timestamp

FailureReason (p. 1034)

If the job failed, the reason that it failed.
Type: String
Length Constraints: Maximum length of 1024.

**HumanTaskConfig (p. 1034)**

Configuration information required for human workers to complete a labeling task.

Type: `HumanTaskConfig (p. 1372)` object

**InputConfig (p. 1034)**

Input configuration information for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.

Type: `LabelingJobInputConfig (p. 1414)` object

**JobReferenceCode (p. 1034)**

A unique identifier for work done as part of a labeling job.

Type: String
Length Constraints: Minimum length of 1.
Pattern: .+

**LabelAttributeName (p. 1034)**

The attribute used as the label in the output manifest file.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

**LabelCategoryConfigS3Uri (p. 1034)**

The S3 location of the JSON file that defines the categories used to label data objects. Please note the following label-category limits:
- Semantic segmentation labeling jobs using automated labeling: 20 labels
- Box bounding labeling jobs (all): 10 labels

The file is a JSON structure in the following format:

```json
{
    "document-version": "2018-11-28"
    "labels": [
    {
        "label": "label 1"
    },
    {
        "label": "label 2"
    },
    ...
```


```
{
  "label": "label n"
}
```

Type: String
Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/?([^/]+)$

**LabelCounters (p. 1034)**

Provides a breakdown of the number of data objects labeled by humans, the number of objects
labeled by machine, the number of objects than couldn't be labeled, and the total number of objects
labeled.

Type: [LabelCounters](#) object

**LabelingJobAlgorithmsConfig (p. 1034)**

Configuration information for automated data labeling.

Type: [LabelingJobAlgorithmsConfig](#) object

**LabelingJobArn (p. 1034)**

The Amazon Resource Name (ARN) of the labeling job.

Type: String
Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*

**LabelingJobName (p. 1034)**

The name assigned to the labeling job when it was created.

Type: String

Pattern: ^[a-zA-Z0-9\-][*-][a-zA-Z0-9\-]*$

**LabelingJobOutput (p. 1034)**

The location of the output produced by the labeling job.

Type: [LabelingJobOutput](#) object

**LabelingJobStatus (p. 1034)**

The processing status of the labeling job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

**LastModifiedTime (p. 1034)**

The date and time that the labeling job was last updated.
OutputConfig (p. 1034)

The location of the job's output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.

Type: LabelingJobOutputConfig (p. 1416) object

RoleArn (p. 1034)

The Amazon Resource Name (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_]+$

StoppingConditions (p. 1034)

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped.

Type: LabelingJobStoppingConditions (p. 1419) object

Tags (p. 1034)

An array of key/value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeModel
Service: Amazon SageMaker Service

Describes a model that you created using the CreateModel API.

Request Syntax

```json
{
  "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ModelName (p. 1040)**

The name of the model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: /^[a-zA-Z0-9-]*[a-zA-Z0-9-]+$*

Required: Yes

Response Syntax

```json
{
  "Containers": [ 
    {
      "ContainerHostname": "string",
      "Environment": {
        "string": "string"
      },
      "Image": "string",
      "Mode": "string",
      "ModelDataUrl": "string",
      "ModelPackageName": "string"
    }
  ],
  "CreationTime": number,
  "EnableNetworkIsolation": boolean,
  "ExecutionRoleArn": "string",
  "ModelArn": "string",
  "ModelName": "string",
  "PrimaryContainer": {
    "ContainerHostname": "string",
    "Environment": {
      "string": "string"
    },
    "Image": "string",
    "Mode": "string",
    "ModelDataUrl": "string",
    "ModelPackageName": "string"
  }
}
```
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Containers (p. 1040)

The containers in the inference pipeline.

Type: Array of ContainerDefinition (p. 1321) objects

Array Members: Maximum number of 5 items.

CreationTime (p. 1040)

A timestamp that shows when the model was created.

Type: Timestamp

EnableNetworkIsolation (p. 1040)

If True, no inbound or outbound network calls can be made to or from the model container.

Type: Boolean

ExecutionRoleArn (p. 1040)

The Amazon Resource Name (ARN) of the IAM role that you specified for the model.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9-\_\+=,.@\-_\/]+$

ModelArn (p. 1040)

The Amazon Resource Name (ARN) of the model.

Type: String


Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]\{12\}:model/.*

ModelName (p. 1040)

Name of the Amazon SageMaker model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\_\+=,.@\-_\/]+$

PrimaryContainer (p. 1040)

The location of the primary inference code, associated artifacts, and custom environment map that the inference code uses when it is deployed in production.
Type: `ContainerDefinition (p. 1321)` object

**VpcConfig (p. 1040)**

A `VpcConfig (p. 1577)` object that specifies the VPC that this model has access to. For more information, see *Protect Endpoints by Using an Amazon Virtual Private Cloud*.

Type: `VpcConfig (p. 1577)` object

**Errors**

For information about the errors that are common to all actions, see *Common Errors (p. 1579)*.

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeModelPackage
Service: Amazon SageMaker Service

Returns a description of the specified model package, which is used to create Amazon SageMaker models or list them on AWS Marketplace.

To create models in Amazon SageMaker, buyers can subscribe to model packages listed on AWS Marketplace.

Request Syntax

```json
{
   "ModelPackageName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ModelPackageName (p. 1043)**

The name of the model package to describe.

Type: String


Pattern: (arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:[a-zA-Z\-]*\/)?([a-zA-Z0-9\-]*[0-9](\[a-zA-Z0-9\-\]\{0,62\})(?!-)$

Required: Yes

Response Syntax

```json
{
   "CertifyForMarketplace": boolean,
   "CreationTime": number,
   "InferenceSpecification": {
      "Containers": [
         {
            "ContainerHostname": "string",
            "Image": "string",
            "ImageDigest": "string",
            "ModelDataUrl": "string",
            "ProductId": "string"
         }
      ],
      "SupportedContentTypes": [ "string" ],
      "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
      "SupportedResponseMIMETypes": [ "string" ],
      "SupportedTransformInstanceTypes": [ "string" ]
   },
   "ModelPackageArn": "string",
   "ModelPackageDescription": "string",
   "ModelPackageName": "string",
   "ModelPackageStatus": "string",
   "ModelPackageStatusDetails": {
      "ImageScanStatuses": [ 
```
{  
  "FailureReason": "string",
  "Name": "string",
  "Status": "string"
},
"ValidationStatuses": [
  {
    "FailureReason": "string",
    "Name": "string",
    "Status": "string"
  }
],
"SourceAlgorithmSpecification": {
  "SourceAlgorithms": [
    {
      "AlgorithmName": "string",
      "ModelDataUrl": "string"
    }
  ]
},
"ValidationSpecification": {
  "ValidationProfiles": [
    {
      "ProfileName": "string",
      "TransformJobDefinition": {
        "BatchStrategy": "string",
        "Environment": {
          "string": "string"
        },
        "MaxConcurrentTransforms": number,
        "MaxPayloadInMB": number,
        "TransformInput": {
          "CompressionType": "string",
          "ContentType": "string",
          "DataSource": {
            "S3DataSource": {
              "S3DataType": "string",
              "S3Uri": "string"
            }
          },
          "SplitType": "string"
        },
        "TransformOutput": {
          "Accept": "string",
          "AssembleWith": "string",
          "KmsKeyId": "string",
          "S3OutputPath": "string"
        },
        "TransformResources": {
          "InstanceCount": number,
          "InstanceType": "string",
          "VolumeKmsKeyId": "string"
        }
      }
    }
  ],
  "ValidationRole": "string"
}

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CertifyForMarketplace** (p. 1043)

Whether the model package is certified for listing on AWS Marketplace.

Type: Boolean

**CreationTime** (p. 1043)

A timestamp specifying when the model package was created.

Type: Timestamp

**InferenceSpecification** (p. 1043)

Details about inference jobs that can be run with models based on this model package.

Type: InferenceSpecification (p. 1396) object

**ModelPackageArn** (p. 1043)

The Amazon Resource Name (ARN) of the model package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*`

**ModelPackageDescription** (p. 1043)

A brief summary of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*`

**ModelPackageName** (p. 1043)

The name of the model package being described.

Type: String


Pattern: `^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$`

**ModelPackageStatus** (p. 1043)

The current status of the model package.

Type: String

Valid Values: Pending | InProgress | Completed | Failed | Deleting

**ModelPackageStatusDetails** (p. 1043)

Details about the current status of the model package.

Type: ModelPackageStatusDetails (p. 1429) object

**SourceAlgorithmSpecification** (p. 1043)

Details about the algorithm that was used to create the model package.
Type: **SourceAlgorithmSpecification** (p. 1512) object

**ValidationSpecification** (p. 1043)

Configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.

Type: **ModelPackageValidationSpecification** (p. 1434) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeMonitoringSchedule
Service: Amazon SageMaker Service

Describes the schedule for a monitoring job.

Request Syntax

```
{
  "MonitoringScheduleName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**MonitoringScheduleName (p. 1047)**

Name of a previously created monitoring schedule.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Syntax

```
{
  "CreationTime": number,
  "EndpointName": "string",
  "FailureReason": "string",
  "LastModifiedTime": number,
  "LastMonitoringExecutionSummary": {
    "CreationTime": number,
    "EndpointName": "string",
    "FailureReason": "string",
    "LastModifiedTime": number,
    "MonitoringExecutionStatus": "string",
    "MonitoringScheduleName": "string",
    "ProcessingJobArn": "string",
    "ScheduledTime": number
  },
  "MonitoringScheduleArn": "string",
  "MonitoringScheduleConfig": {
    "MonitoringJobDefinition": {
      "BaselineConfig": {
        "ConstraintsResource": {
          "S3Uri": "string"
        },
        "StatisticsResource": {
          "S3Uri": "string"
        }
      }
    }
  }
}
```
"Environment": { 
  "string": "string"
},
"MonitoringAppSpecification": { 
  "ContainerArguments": [ "string" ],
  "ContainerEntrypoint": [ "string" ],
  "ImageUri": "string",
  "PostAnalyticsProcessorSourceUri": "string",
  "RecordPreprocessorSourceUri": "string"
},
"MonitoringInputs": [ 
  { 
    "EndpointInput": { 
      "EndpointName": "string",
      "LocalPath": "string",
      "S3DataDistributionType": "string",
      "S3InputMode": "string"
    }
  }
],
"MonitoringOutputConfig": { 
  "KmsKeyId": "string",
  "MonitoringOutputs": [ 
    { 
      "S3Output": { 
        "LocalPath": "string",
        "S3UploadMode": "string",
        "S3Uri": "string"
      }
    }
  ]
},
"MonitoringResources": { 
  "ClusterConfig": { 
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "NetworkConfig": { 
    "EnableNetworkIsolation": boolean,
    "VpcConfig": { 
      "SecurityGroupIds": [ "string" ],
      "Subnets": [ "string" ]
    }
  },
  "RoleArn": "string",
  "StoppingCondition": { 
    "MaxRuntimeInSeconds": number
  },
  "ScheduleConfig": { 
    "ScheduleExpression": "string"
  }
},
"MonitoringScheduleName": "string",
"MonitoringScheduleStatus": "string"

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
**CreationTime (p. 1047)**

The time at which the monitoring job was created.

Type: Timestamp

**EndpointName (p. 1047)**

The name of the endpoint for the monitoring job.

Type: String

Length Constraints: Maximum length of 63.

Pattern: \^[a-zA-Z0-9](-*[a-zA-Z0-9])*\$

**FailureReason (p. 1047)**

A string, up to one KB in size, that contains the reason a monitoring job failed, if it failed.

Type: String

Length Constraints: Maximum length of 1024.

**LastModifiedTime (p. 1047)**

The time at which the monitoring job was last modified.

Type: Timestamp

**LastMonitoringExecutionSummary (p. 1047)**

Describes metadata on the last execution to run, if there was one.

Type: MonitoringExecutionSummary (p. 1442) object

**MonitoringScheduleArn (p. 1047)**

The Amazon Resource Name (ARN) of the monitoring schedule.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

**MonitoringScheduleConfig (p. 1047)**

The configuration object that specifies the monitoring schedule and defines the monitoring job.

Type: MonitoringScheduleConfig (p. 1451) object

**MonitoringScheduleName (p. 1047)**

Name of the monitoring schedule.

Type: String


Pattern: \^[a-zA-Z0-9](-*[a-zA-Z0-9])*\$

**MonitoringScheduleStatus (p. 1047)**

The status of an monitoring job.

Type: String
Valid Values: Pending | Failed | Scheduled | Stopped

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound
Resource being access is not found.
HTTP Status Code: 400

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeNotebookInstance
Service: Amazon SageMaker Service
Returns information about a notebook instance.

Request Syntax

```json
{
    "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 1051)**

The name of the notebook instance that you want information about.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

Response Syntax

```json
{
    "AcceleratorTypes": [ "string" ],
    "AdditionalCodeRepositories": [ "string" ],
    "CreationTime": number,
    "DefaultCodeRepository": "string",
    "DirectInternetAccess": "string",
    "FailureReason": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LastModifiedTime": number,
    "NetworkInterfaceId": "string",
    "NotebookInstanceArn": "string",
    "NotebookInstanceLifecycleConfigName": "string",
    "NotebookInstanceName": "string",
    "NotebookInstanceStatus": "string",
    "RoleArn": "string",
    "RootAccess": "string",
    "SecurityGroups": [ "string" ],
    "SubnetId": "string",
    "Url": "string",
    "VolumeSizeInGB": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**AcceleratorTypes (p. 1051)**

A list of the Elastic Inference (EI) instance types associated with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values:

- ml.eia1.medium
- ml.eia1.large
- ml.eia1.xlarge
- ml.eia2.medium
- ml.eia2.large
- ml.eia2.xlarge

**AdditionalCodeRepositories (p. 1051)**

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.

Pattern: ^https://(\[^/]+)/(.*$)|^\[a-zA-Z0-9\](-*[a-zA-Z0-9]*)$

**CreationTime (p. 1051)**

A timestamp. Use this parameter to return the time when the notebook instance was created

Type: Timestamp

**DefaultCodeRepository (p. 1051)**

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String


Pattern: ^https://((\^[^/]+)/(.*$)|^[a-zA-Z0-9\](-*[a-zA-Z0-9]*)$

**DirectInternetAccess (p. 1051)**

Describes whether Amazon SageMaker provides internet access to the notebook instance. If this value is set to Disabled, the notebook instance does not have internet access, and cannot connect to Amazon SageMaker training and endpoint services.

For more information, see Notebook Instances Are Internet-Enabled by Default.

Type: String

Valid Values: Enabled | Disabled

**FailureReason (p. 1051)**

If status is Failed, the reason it failed.
Type: String

Length Constraints: Maximum length of 1024.

**InstanceType (p. 1051)**

The type of ML compute instance running on the notebook instance.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge |
ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge |
ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge |
ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge |
ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge |
ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge |
ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge

**KmsKeyId (p. 1051)**

The AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

**LastModifiedTime (p. 1051)**

A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.

Type: Timestamp

**NetworkInterfaceId (p. 1051)**

The network interface IDs that Amazon SageMaker created at the time of creating the instance.

Type: String

**NotebookInstanceArn (p. 1051)**

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

**NotebookInstanceLifecycleConfigName (p. 1051)**

Returns the name of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9]-*[a-zA-Z0-9]*

**NotebookInstanceName (p. 1051)**

The name of the Amazon SageMaker notebook instance.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

NotebookInstanceStatus (p. 1051)
The status of the notebook instance.
Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating

RoleArn (p. 1051)
The Amazon Resource Name (ARN) of the IAM role associated with the instance.
Type: String
Pattern: ^arn:aws[a-zA-Z]*:iam::\d{12}:role/\?\[a-zA-ZA-Z_0-9=+.@\-_/]+$

RootAccess (p. 1051)
Whether root access is enabled or disabled for users of the notebook instance.

Note
Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Type: String
Valid Values: Enabled | Disabled

SecurityGroups (p. 1051)
The IDs of the VPC security groups.
Type: Array of strings
Array Members: Maximum number of 5 items.
Length Constraints: Maximum length of 32.
Pattern: [-0-9a-zA-Z]+

SubnetId (p. 1051)
The ID of the VPC subnet.
Type: String
Length Constraints: Maximum length of 32.
Pattern: [-0-9a-zA-Z]+

Url (p. 1051)
The URL that you use to connect to the Jupyter notebook that is running in your notebook instance.
Type: String
VolumeSizeInGB (p. 1051)

The size, in GB, of the ML storage volume attached to the notebook instance.

Type: Integer


Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeNotebookInstanceLifecycleConfig
Service: Amazon SageMaker Service

Returns a description of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```json
{
   "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 1056)**

The name of the lifecycle configuration to describe.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 

Required: Yes

Response Syntax

```json
{
   "CreationTime": number,
   "LastModifiedTime": number,
   "NotebookInstanceLifecycleConfigArn": "string",
   "NotebookInstanceLifecycleConfigName": "string",
   "OnCreate": [
      { "Content": "string" }
   ],
   "OnStart": [
      { "Content": "string" }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
CreationTime (p. 1056)
A timestamp that tells when the lifecycle configuration was created.
Type: Timestamp

LastModifiedTime (p. 1056)
A timestamp that tells when the lifecycle configuration was last modified.
Type: Timestamp

NotebookInstanceLifecycleConfigArn (p. 1056)
The Amazon Resource Name (ARN) of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 1056)
The name of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 63.
Pattern: \^[a-zA-Z0-9](\-*[a-zA-Z0-9])*\n
OnCreate (p. 1056)
The shell script that runs only once, when you create a notebook instance.
Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects
Array Members: Maximum number of 1 item.

OnStart (p. 1056)
The shell script that runs every time you start a notebook instance, including when you create the notebook instance.
Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects
Array Members: Maximum number of 1 item.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeProcessingJob
Service: Amazon SageMaker Service

Returns a description of a processing job.

Request Syntax

```json
{
    "ProcessingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

ProcessingJobName (p. 1059)

The name of the processing job. The name must be unique within an AWS Region in the AWS account.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Syntax

```json
{
    "AppSpecification": {
        "ContainerArguments": [ "string" ],
        "ContainerEntrypoint": [ "string" ],
        "ImageUri": "string"
    },
    "AutoMLJobArn": "string",
    "CreationTime": number,
    "Environment": {
        "string": "string"
    },
    "ExitMessage": "string",
    "ExperimentConfig": {
        "ExperimentName": "string",
        "TrialComponentDisplayName": "string",
        "TrialName": "string"
    },
    "FailureReason": "string",
    "LastModifiedTime": number,
    "MonitoringScheduleArn": "string",
    "NetworkConfig": {
        "EnableNetworkIsolation": boolean,
        "VpcConfig": {
            "SecurityGroupId": [ "string" ],
            "Subnets": [ "string" ]
        }
    }
}
```
"ProcessingEndTime": number,
"ProcessingInputs": [
  {
    "InputName": "string",
    "S3Input": {
      "LocalPath": "string",
      "S3CompressionType": "string",
      "S3DataDistributionType": "string",
      "S3DataType": "string",
      "S3InputMode": "string",
      "S3Uri": "string"
    }
  }
],
"ProcessingJobArn": "string",
"ProcessingJobName": "string",
"ProcessingJobStatus": "string",
"ProcessingOutputConfig": {
  "KmsKeyId": "string",
  "Outputs": [
    {
      "OutputName": "string",
      "S3Output": {
        "LocalPath": "string",
        "S3UploadMode": "string",
        "S3Uri": "string"
      }
    }
  ]
},
"ProcessingResources": {
  "ClusterConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  }
},
"ProcessingStartTime": number,
"RoleArn": "string",
"StoppingCondition": {
  "MaxRuntimeInSeconds": number
},
"TrainingJobArn": "string"

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AppSpecification (p. 1059)

Configures the processing job to run a specified container image.

Type: AppSpecification (p. 1290) object

AutoMLJobArn (p. 1059)

The ARN of an AutoML job associated with this processing job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.
Pattern: arn:aws*[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:automl-job/.*

**CreationTime (p. 1059)**

The time at which the processing job was created.

Type: Timestamp

**Environment (p. 1059)**

The environment variables set in the Docker container.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: \[a-zA-Z_\][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 256.

Value Pattern: \S\s*

**ExitMessage (p. 1059)**

An optional string, up to one KB in size, that contains metadata from the processing container when the processing job exits.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: \S\s*

**ExperimentConfig (p. 1059)**

The configuration information used to create an experiment.

Type: ExperimentConfig (p. 1348) object

**FailureReason (p. 1059)**

A string, up to one KB in size, that contains the reason a processing job failed, if it failed.

Type: String

Length Constraints: Maximum length of 1024.

**LastModifiedTime (p. 1059)**

The time at which the processing job was last modified.

Type: Timestamp

**MonitoringScheduleArn (p. 1059)**

The ARN of a monitoring schedule for an endpoint associated with this processing job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

**NetworkConfig (p. 1059)**

Networking options for a processing job.

Type: NetworkConfig (p. 1457) object
ProcessingEndTime (p. 1059)

The time at which the processing job completed.

Type: Timestamp

ProcessingInputs (p. 1059)

The inputs for a processing job.

Type: Array of ProcessingInput (p. 1474) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

ProcessingJobArn (p. 1059)

The Amazon Resource Name (ARN) of the processing job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:processing-job/.*

ProcessingJobName (p. 1059)

The name of the processing job. The name must be unique within an AWS Region in the AWS account.

Type: String


Pattern: ^[a-zA-Z0-9\-]*(\-[a-zA-Z0-9\-]*)*

ProcessingJobStatus (p. 1059)

Provides the status of a processing job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

ProcessingOutputConfig (p. 1059)

Output configuration for the processing job.

Type: ProcessingOutputConfig (p. 1478) object

ProcessingResources (p. 1059)

Identifies the resources, ML compute instances, and ML storage volumes to deploy for a processing job. In distributed training, you specify more than one instance.

Type: ProcessingResources (p. 1479) object

ProcessingStartTime (p. 1059)

The time at which the processing job started.

Type: Timestamp

RoleArn (p. 1059)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

Type: String

Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9=,\.@\-_]+$  

**StoppingCondition (p. 1059)**

The time limit for how long the processing job is allowed to run.

Type: ProcessingStoppingCondition (p. 1483) object

**TrainingJobArn (p. 1059)**

The ARN of a training job associated with this processing job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeSubscribedWorkteam
Service: Amazon SageMaker Service

Gets information about a work team provided by a vendor. It returns details about the subscription with a vendor in the AWS Marketplace.

Request Syntax

```json
{
   "WorkteamArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

WorkteamArn (p. 1064)

The Amazon Resource Name (ARN) of the subscribed work team to describe.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:workteam/.*`

Required: Yes

Response Syntax

```json
{
   "SubscribedWorkteam": {
      "ListingId": "string",
      "MarketplaceDescription": "string",
      "MarketplaceTitle": "string",
      "SellerName": "string",
      "WorkteamArn": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

SubscribedWorkteam (p. 1064)

A `Workteam` instance that contains information about the work team.

Type: `SubscribedWorkteam` (p. 1514) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTrainingJob
Service: Amazon SageMaker Service

Returns information about a training job.

Request Syntax

```json
{
    "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrainingJobName (p. 1066)**

The name of the training job.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Syntax

```json
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "EnableSageMakerMetricsTimeSeries": boolean,
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "AutoMLJobArn": "string",
    "BillableTimeInSeconds": number,
    "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
    },
    "CreationTime": number,
    "DebugHookConfig": {
        "CollectionConfigurations": [
            {
                "CollectionName": "string",
                "CollectionParameters": {
                    "string": "string"
                }
            }
        ],
```
"HookParameters": {
    "string": "string"
  },
"LocalPath": "string",
"S3OutputPath": "string"
},
"DebugRuleConfigurations": [
  {
    "InstanceType": "string",
    "LocalPath": "string",
    "RuleConfigurationName": "string",
    "RuleEvaluatorImage": "string",
    "RuleParameters": {
      "string": "string"
    },
    "S3OutputPath": "string",
    "VolumeSizeInGB": number
  }
],
"DebugRuleEvaluationStatuses": [
  {
    "LastModifiedTime": number,
    "RuleConfigurationName": "string",
    "RuleEvaluationJobArn": "string",
    "RuleEvaluationStatus": "string",
    "StatusDetails": "string"
  }
],
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"ExperimentConfig": {
    "ExperimentName": "string",
    "TrialComponentDisplayName": "string",
    "TrialName": "string"
},
"FailureReason": "string",
"FinalMetricDataList": [
  {
    "MetricName": "string",
    "Timestamp": number,
    "Value": number
  }
],
"HyperParameters": {
    "string": "string"
},
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    }
  }
]
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
  "Seed": number
}
],
"LabelingJobArn": "string",
"LastModifiedTime": number,
"ModelArtifacts": {
  "S3ModelArtifacts": "string"
},
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"SecondaryStatus": "string",
"SecondaryStatusTransitions": [
  {
    "EndTime": number,
    "StartTime": number,
    "Status": "string",
    "StatusMessage": "string"
  }
],
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},
"TrainingEndTime": number,
"TrainingJobArn": "string",
"TrainingJobName": "string",
"TrainingJobStatus": "string",
"TrainingStartTime": number,
"TrainingTimeInSeconds": number,
"TuningJobArn": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}
}

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AlgorithmSpecification (p. 1066)**

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 1274) object
AutoMLJobArn (p. 1066)

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:automl-job/.*`

BillableTimeInSeconds (p. 1066)

The billable time in seconds.

You can calculate the savings from using managed spot training using the formula \(1 - \frac{\text{BillableTimeInSeconds}}{\text{TrainingTimeInSeconds}} \times 100\). For example, if BillableTimeInSeconds is 100 and TrainingTimeInSeconds is 500, the savings is 80%.

Type: Integer

Valid Range: Minimum value of 1.

CheckpointConfig (p. 1066)

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

CreationTime (p. 1066)

A timestamp that indicates when the training job was created.

Type: Timestamp

DebugHookConfig (p. 1066)

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

DebugRuleConfigurations (p. 1066)

Configuration information for debugging rules.

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

DebugRuleEvaluationStatuses (p. 1066)

Status about the debug rule evaluation.

Type: Array of DebugRuleEvaluationStatus (p. 1335) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

EnableInterContainerTrafficEncryption (p. 1066)

To encrypt all communications between ML compute instances in distributed training, choose `true`. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithms in distributed training.

Type: Boolean

EnableManagedSpotTraining (p. 1066)

A Boolean indicating whether managed spot training is enabled (`true`) or not (`false`).
Type: Boolean

**EnableNetworkIsolation (p. 1066)**

If you want to allow inbound or outbound network calls, except for calls between peers within a training cluster for distributed training, choose `true`. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean

**ExperimentConfig (p. 1066)**

Configuration for the experiment.

Type: `ExperimentConfig (p. 1348)` object

**FailureReason (p. 1066)**

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**FinalMetricDataList (p. 1066)**

A collection of `MetricData` objects that specify the names, values, and dates and times that the training algorithm emitted to Amazon CloudWatch.

Type: Array of `MetricData (p. 1424)` objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

**HyperParameters (p. 1066)**

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: `.\*`

Value Length Constraints: Maximum length of 256.

Value Pattern: `.\*`

**InputDataConfig (p. 1066)**

An array of `Channel` objects that describes each data input channel.

Type: Array of `Channel (p. 1310)` objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

**LabelingJobArn (p. 1066)**

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`
**LastModifiedTime (p. 1066)**

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

**ModelArtifacts (p. 1066)**

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 1426) object

**OutputDataConfig (p. 1066)**

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: OutputDataConfig (p. 1466) object

**ResourceConfig (p. 1066)**

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: ResourceConfig (p. 1496) object

**RoleArn (p. 1066)**

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/\?([a-zA-Z_0-9+=,.@\-_/]+)+$`

**SecondaryStatus (p. 1066)**

Provides detailed information about the state of the training job. For detailed information on the secondary status of the training job, see StatusMessage under SecondaryStatusTransition (p. 1507).

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

- **InProgress**
  - Starting - Starting the training job.
  - Downloading - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
  - Training - Training is in progress.
  - Interrupted - The job stopped because the managed spot training instances were interrupted.
  - Uploading - Training is complete and the model artifacts are being uploaded to the S3 location.

- **Completed**
  - Completed - The training job has completed.

- **Failed**
  - Failed - The training job has failed. The reason for the failure is returned in the FailureReason field of DescribeTrainingJobResponse.

- **Stopped**
  - MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed runtime.
- MaxWaitTimeExceeded - The job stopped because it exceeded the maximum allowed wait time.
- Stopped - The training job has stopped.

## Stopping
- Stopping - Stopping the training job.

**Important**
Valid values for `SecondaryStatus` are subject to change.

We no longer support the following secondary statuses:
- LaunchingMLInstances
- PreparingTrainingStack
- DownloadingTrainingImage

Type: String

Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded

### SecondaryStatusTransitions (p. 1066)
A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of `SecondaryStatusTransition (p. 1507)` objects

### StoppingCondition (p. 1066)
Specifies a limit to how long a model training job can run. It also specifies the maximum time to wait for a spot instance. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: `StoppingCondition (p. 1513)` object

### TensorBoardOutputConfig (p. 1066)
Configuration of storage locations for TensorBoard output.

Type: `TensorBoardOutputConfig (p. 1519)` object

### TrainingEndTime (p. 1066)
Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of `TrainingStartTime` and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: `Timestamp`

### TrainingJobArn (p. 1066)
The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`
TrainingJobName (p. 1066)

Name of the model training job.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

TrainingJobStatus (p. 1066)

The status of the training job.

Amazon SageMaker provides the following training job statuses:

- InProgress - The training is in progress.
- Completed - The training job has completed.
- Failed - The training job has failed. To see the reason for the failure, see the FailureReason field in the response to a DescribeTrainingJobResponse call.
- Stopping - The training job is stopping.
- Stopped - The training job has stopped.

For more detailed information, see SecondaryStatus.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

TrainingStartTime (p. 1066)

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of TrainingEndTime. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

TrainingTimeInSeconds (p. 1066)

The training time in seconds.

Type: Integer

Valid Range: Minimum value of 1.

TuningJobArn (p. 1066)

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

VpcConfig (p. 1066)

A VpcConfig (p. 1577) object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTransformJob
Service: Amazon SageMaker Service

Returns information about a transform job.

Request Syntax

{
    "TransformJobName": "string"
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

TransformJobName (p. 1075)

The name of the transform job that you want to view details of.

Type: String
Pattern: ^[a-zA-Z0-9\-\[a-zA-Z0-9\]*\[a-zA-Z0-9\]]*$
Required: Yes

Response Syntax

{
    "AutoMLJobArn": "string",
    "BatchStrategy": "string",
    "CreationTime": number,
    "DataProcessing": {
        "InputFilter": "string",
        "JoinSource": "string",
        "OutputFilter": "string"
    },
    "Environment": {
        "string": "string"
    },
    "ExperimentConfig": {
        "ExperimentName": "string",
        "TrialComponentDisplayName": "string",
        "TrialName": "string"
    },
    "FailureReason": "string",
    "LabelingJobArn": "string",
    "MaxConcurrentTransforms": number,
    "MaxPayloadInMB": number,
    "ModelName": "string",
    "TransformEndTime": number,
    "TransformInput": {
        "CompressionType": "string",
        "ContentType": "string",
        "DataSource": {
            "S3DataSource": {
            ...
        
1075
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AutoMLJobArn (p. 1075)
Type: String
Length Constraints: Minimum length of 1. Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:automl-job/.*

BatchStrategy (p. 1075)
Specifies the number of records to include in a mini-batch for an HTTP inference request. A record is a single unit of input data that inference can be made on. For example, a single line in a CSV file is a record.

To enable the batch strategy, you must set SplitType to Line, RecordIO, or TFRecord.
Type: String
Valid Values: MultiRecord | SingleRecord

CreationTime (p. 1075)
A timestamp that shows when the transform Job was created.
Type: Timestamp

DataProcessing (p. 1075)
The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.
**Environment (p. 1075)**

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: `[a-zA-Z_]\[a-zA-Z0-9_]`*

Value Length Constraints: Maximum length of 10240.

Value Pattern: `[^\s]*`

**ExperimentConfig (p. 1075)**

Configuration for the experiment.

Type: `ExperimentConfig (p. 1348)` object

**FailureReason (p. 1075)**

If the transform job failed, `FailureReason` describes why it failed. A transform job creates a log file, which includes error messages, and stores it as an Amazon S3 object. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch.

Type: String

Length Constraints: Maximum length of 1024.

**LabelingJobArn (p. 1075)**

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-zA-Z-]*:sagemaker:[a-zA-Z0-9-]*:[0-9]{12}:labeling-job/.*`

**MaxConcurrentTransforms (p. 1075)**

The maximum number of parallel requests on each instance node that can be launched in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

**MaxPayloadInMB (p. 1075)**

The maximum payload size, in MB, used in the transform job.

Type: Integer

Valid Range: Minimum value of 0.

**ModelName (p. 1075)**

The name of the model used in the transform job.

Type: String
Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

**TransformEndTime (p. 1075)**

Indicates when the transform job has been completed, or has stopped or failed. You are billed for the time interval between this time and the value of `TransformStartTime`.

Type: Timestamp

**TransformInput (p. 1075)**

Describes the dataset to be transformed and the Amazon S3 location where it is stored.

Type: `TransformInput (p. 1536)` object

**TransformJobArn (p. 1075)**

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`

**TransformJobName (p. 1075)**

The name of the transform job.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

**TransformJobStatus (p. 1075)**

The status of the transform job. If the transform job failed, the reason is returned in the `FailureReason` field.

Type: String

Valid Values: `InProgress` | `Completed` | `Failed` | `Stopping` | `Stopped`

**TransformOutput (p. 1075)**

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: `TransformOutput (p. 1542)` object

**TransformResources (p. 1075)**

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: `TransformResources (p. 1544)` object

**TransformStartTime (p. 1075)**

Indicates when the transform job starts on ML instances. You are billed for the time interval between this time and the value of `TransformEndTime`.

Type: Timestamp
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTrial
Service: Amazon SageMaker Service

Provides a list of a trial's properties.

Request Syntax

```json
{
   "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialName (p. 1080)**

The name of the trial to describe.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```json
{
   "CreatedBy": {
      "DomainId": "string",
      "UserProfileArn": "string",
      "UserProfileName": "string"
   },
   "CreationTime": number,
   "DisplayName": "string",
   "ExperimentName": "string",
   "LastModifiedBy": {
      "DomainId": "string",
      "UserProfileArn": "string",
      "UserProfileName": "string"
   },
   "LastModifiedTime": number,
   "Source": {
      "SourceArn": "string",
      "SourceType": "string"
   },
   "TrialArn": "string",
   "TrialName": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CreatedBy (p. 1080)**
Who created the trial.
Type: UserContext (p. 1572) object

**CreationTime (p. 1080)**
When the trial was created.
Type: Timestamp

**DisplayName (p. 1080)**
The name of the trial as displayed. If DisplayName isn't specified, TrialName is displayed.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

**ExperimentName (p. 1080)**
The name of the experiment the trial is part of.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

**LastModifiedBy (p. 1080)**
Who last modified the trial.
Type: UserContext (p. 1572) object

**LastModifiedTime (p. 1080)**
When the trial was last modified.
Type: Timestamp

**Source (p. 1080)**
The Amazon Resource Name (ARN) of the source and, optionally, the job type.
Type: TrialSource (p. 1564) object

**TrialArn (p. 1080)**
The Amazon Resource Name (ARN) of the trial.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:experiment-trial/.*

**TrialName (p. 1080)**
The name of the trial.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9]+(-*[a-zA-Z0-9])*$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTrialComponent
Service: Amazon SageMaker Service
Provides a list of a trials component's properties.

Request Syntax
```json
{
   "TrialComponentName": "string"
}
```

Request Parameters
For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialComponentName (p. 1083)**

The name of the trial component to describe.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* ^[

Required: Yes

Response Syntax
```json
{
   "CreatedBy": {
       "DomainId": "string",
       "UserProfileArn": "string",
       "UserProfileName": "string"
   },
   "CreationTime": number,
   "DisplayName": "string",
   "EndTime": number,
   "InputArtifacts": {
       "string": {
           "MediaType": "string",
           "Value": "string"
       }
   },
   "LastModifiedBy": {
       "DomainId": "string",
       "UserProfileArn": "string",
       "UserProfileName": "string"
   },
   "LastModifiedTime": number,
   "Metrics": [
       {
           "Avg": number,
           "Count": number,
           "Last": number,
           "Max": number,
           "MetricName": "string"
       }
   ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreatedBy (p. 1083)

Who created the component.

Type: UserContext (p. 1572) object

CreationTime (p. 1083)

When the component was created.

Type: Timestamp

DisplayName (p. 1083)

The name of the component as displayed. If DisplayName isn't specified, TrialComponentName is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9]+(-*[a-zA-Z0-9]+)*

EndTime (p. 1083)

When the component ended.

Type: Timestamp
**InputArtifacts (p. 1083)**

The input artifacts of the component.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: . *

**LastModifiedBy (p. 1083)**

Who last modified the component.

Type: `UserContext (p. 1572)` object

**LastModifiedTime (p. 1083)**

When the component was last modified.

Type: Timestamp

**Metrics (p. 1083)**

The metrics for the component.

Type: Array of `TrialComponentMetricSummary (p. 1554)` objects

**OutputArtifacts (p. 1083)**

The output artifacts of the component.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: . *

**Parameters (p. 1083)**

The hyperparameters of the component.

Type: String to `TrialComponentParameterValue (p. 1556)` object map

Key Length Constraints: Maximum length of 256.

Key Pattern: . *

**Source (p. 1083)**

The Amazon Resource Name (ARN) of the source and, optionally, the job type.

Type: `TrialComponentSource (p. 1559)` object

**StartTime (p. 1083)**

When the component started.

Type: Timestamp

**Status (p. 1083)**

The status of the component. States include:

- InProgress
- Completed
- Failed
Type: TrialComponentStatus (p. 1561) object

**TrialComponentArn (p. 1083)**

The Amazon Resource Name (ARN) of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*`

**TrialComponentName (p. 1083)**

The name of the trial component.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](\-*[a-zA-Z0-9])*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeUserProfile

Service: Amazon SageMaker Service

Describes the user profile.

Request Syntax

```json
{
  "DomainId": "string",
  "UserProfileName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 1087)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

**UserProfileName (p. 1087)**

The user profile name.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Syntax

```json
{
  "CreationTime": number,
  "DomainId": "string",
  "FailureReason": "string",
  "HomeEfsFileSystemUid": "string",
  "LastModifiedTime": number,
  "SingleSignOnUserIdentifier": "string",
  "SingleSignOnUserValue": "string",
  "Status": "string",
  "UserProfileArn": "string",
  "UserProfileName": "string",
  "UserSettings": {
    "ExecutionRole": "string",
    "JupyterServerAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    }
  }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CreationTime (p. 1087)**

The creation time.

Type: Timestamp

**DomainId (p. 1087)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

**FailureReason (p. 1087)**

The failure reason.

Type: String

Length Constraints: Maximum length of 1024.

**HomeEfsFileSystemUid (p. 1087)**

The homa Amazon Elastic File System (EFS)Uid.

Type: String

Length Constraints: Maximum length of 10.

Pattern: \d+

**LastModifiedTime (p. 1087)**

The last modified time.

Type: Timestamp
**SingleSignOnUserIdentifier (p. 1087)**

The SSO user identifier.

Type: String

Pattern: UserName

**SingleSignOnUserValue (p. 1087)**

The SSO user value.

Type: String

Length Constraints: Maximum length of 256.

**Status (p. 1087)**

The status.

Type: String

Valid Values: Deleting | Failed | InService | Pending

**UserProfileArn (p. 1087)**

The user profile Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:user-profile/.*

**UserProfileName (p. 1087)**

The user profile name.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9]{1,63}(-*[a-zA-Z0-9]{0,62})*

**UserSettings (p. 1087)**

A collection of settings.

Type: UserSettings (p. 1575) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeWorkteam
Service: Amazon SageMaker Service

Gets information about a specific work team. You can see information such as the create date, the last updated date, membership information, and the work team's Amazon Resource Name (ARN).

Request Syntax

```json
{
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**WorkteamName (p. 1091)**

The name of the work team to return a description of.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```json
{
    "Workteam": {
        "CreateDate": number,
        "Description": "string",
        "LastUpdatedDate": number,
        "MemberDefinitions": [
            {
                "CognitoMemberDefinition": {
                    "ClientId": "string",
                    "UserGroup": "string",
                    "UserPool": "string"
                }
            }
        ],
        "NotificationConfiguration": {
            "NotificationTopicArn": "string"
        },
        "ProductListingIds": [ "string" ],
        "SubDomain": "string",
        "WorkteamArn": "string",
        "WorkteamName": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**Workteam (p. 1091)**

A Workteam instance that contains information about the work team.

Type: Workteam (p. 1578) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DisassociateTrialComponent
Service: Amazon SageMaker Service

Disassociates a trial component from a trial. This doesn't effect other trials the component is associated with. Before you can delete a component, you must disassociate the component from all trials it is associated with. To associate a trial component with a trial, call the AssociateTrialComponent (p. 852) API.

Request Syntax

```json
{
   "TrialComponentName": "string",
   "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**TrialComponentName (p. 1093)**

The name of the component to disassociate from the trial.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**TrialName (p. 1093)**

The name of the trial to disassociate from.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Syntax

```json
{
   "TrialArn": "string",
   "TrialComponentArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
**TrialArn (p. 1093)**

The Amazon Resource Name (ARN) of the trial.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial/.*`

**TrialComponentArn (p. 1093)**

The ARN of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*`

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 1579)](#).

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
GetSearchSuggestions
Service: Amazon SageMaker Service

An auto-complete API for the search functionality in the Amazon SageMaker console. It returns suggestions of possible matches for the property name to use in Search queries. Provides suggestions for HyperParameters, Tags, and Metrics.

Request Syntax

```json
{
  "Resource": "string",
  "SuggestionQuery": {
    "PropertyNameQuery": {
      "PropertyNameHint": "string"
    }
  }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

Resource (p. 1095)

The name of the Amazon SageMaker resource to Search for.

Type: String

Valid Values: TrainingJob | Experiment | ExperimentTrial | ExperimentTrialComponent

Required: Yes

SuggestionQuery (p. 1095)

Limits the property names that are included in the response.

Type: SuggestionQuery (p. 1516) object

Required: No

Response Syntax

```json
{
  "PropertyNameSuggestions": [
    {
      "PropertyName": "string"
    }
  ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
PropertyNameSuggestions (p. 1095)
A list of property names for a Resource that match a SuggestionQuery.
Type: Array of PropertyNameSuggestion (p. 1489) objects

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**ListAlgorithms**  
Service: Amazon SageMaker Service

Lists the machine learning algorithms that have been created.

**Request Syntax**

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1097)**

A filter that returns only algorithms created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1097)**

A filter that returns only algorithms created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 1097)**

The maximum number of algorithms to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1097)**

A string in the algorithm name. This filter returns only algorithms whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-]+

Required: No
NextToken (p. 1097)

If the response to a previous ListAlgorithms request was truncated, the response includes a NextToken. To retrieve the next set of algorithms, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1097)

The parameter by which to sort the results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1097)

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
  "AlgorithmSummaryList": [
    {
      "AlgorithmArn": "string",
      "AlgorithmDescription": "string",
      "AlgorithmName": "string",
      "AlgorithmStatus": "string",
      "CreationTime": number
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmSummaryList (p. 1098)

> An array of AlgorithmSummary objects, each of which lists an algorithm.

Type: Array of AlgorithmSummary (p. 1278) objects

NextToken (p. 1098)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of algorithms, use it in the subsequent request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: . *

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListApps
Service: Amazon SageMaker Service
Lists apps.

Request Syntax

```
{
  "DomainIdEquals": "string",
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "UserProfileNameEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainIdEquals (p. 1100)**

A parameter to search for the domain ID.

*Type:* String

*Length Constraints:* Maximum length of 63.

*Required:* No

**MaxResults (p. 1100)**

Returns a list up to a specified limit.

*Type:* Integer

*Valid Range:* Minimum value of 1. Maximum value of 100.

*Required:* No

**NextToken (p. 1100)**

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

*Type:* String

*Length Constraints:* Maximum length of 8192.

*Pattern:* .*

*Required:* No

**SortBy (p. 1100)**

The parameter by which to sort the results. The default is CreationTime.

*Type:* String

*Valid Values:* CreationTime
Required: No

**SortOrder (p. 1100)**

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**UserProfileNameEquals (p. 1100)**

A parameter to search by user profile name.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

### Response Syntax

```json
{
   "Apps": [
      {
         "AppName": "string",
         "AppType": "string",
         "CreationTime": number,
         "DomainId": "string",
         "Status": "string",
         "UserProfileName": "string"
      }
   ],
   "NextToken": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Apps (p. 1101)**

The list of apps.

Type: Array of [AppDetails (p. 1288)] objects

**NextToken (p. 1101)**

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListAutoMLJobs
Service: Amazon SageMaker Service

Request a list of jobs.

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1103)**

Request a list of jobs, using a filter for time.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1103)**

Request a list of jobs, using a filter for time.

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1103)**

Request a list of jobs, using a filter for time.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1103)**

Request a list of jobs, using a filter for time.

Type: Timestamp

Required: No

**MaxResults (p. 1103)**

Request a list of jobs up to a specified limit.

Type: Integer
NameContains (p. 1103)
Request a list of jobs, using a search filter for name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: \[a-zA-Z0-9\-\]+
Required: No
NextToken (p. 1103)
If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No
SortBy (p. 1103)
The parameter by which to sort the results. The default is AutoMLJobName.
Type: String
Valid Values: Name | CreationTime | Status
Required: No
SortOrder (p. 1103)
The sort order for the results. The default is Descending.
Type: String
Valid Values: Ascending | Descending
Required: No
StatusEquals (p. 1103)
Request a list of jobs, using a filter for status.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: No

Response Syntax
{
  "AutoMLJobSummaries": [
  
}
"AutoMLJobArn": "string",
"AutoMLJobName": "string",
"AutoMLJobSecondaryStatus": "string",
"AutoMLJobStatus": "string",
"CreationTime": number,
"EndTime": number,
"FailureReason": "string",
"LastModifiedTime": number,
"
"
"NextToken": "string"
"

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AutoMLJobSummaries (p. 1104)

Returns a summary list of jobs.

Type: Array of AutoMLJobSummary (p. 1301) objects

NextToken (p. 1104)

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: \.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListCandidatesForAutoMLJob
Service: Amazon SageMaker Service

List the Candidates created for the job.

Request Syntax

```
{
    "AutoMLJobName": "string",
    "CandidateNameEquals": "string",
    "MaxResults": number,
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AutoMLJobName (p. 1106)**

List the Candidates created for the job by providing the job's name.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

**CandidateNameEquals (p. 1106)**

List the Candidates for the job and filter by candidate name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Required: No

**MaxResults (p. 1106)**

List the job's Candidates up to a specified limit.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NextToken (p. 1106)**

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String
Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**SortBy (p. 1106)**

The parameter by which to sort the results. The default is Descending.

Type: String

Valid Values: CreationTime | Status | FinalObjectiveMetricValue

Required: No

**SortOrder (p. 1106)**

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1106)**

List the Candidates for the job and filter by status.

Type: String

Valid Values: Completed | InProgress | Failed | Stopped | Stopping

Required: No

**Response Syntax**

```json
{
  "Candidates": [ 
    { 
      "CandidateName": "string",
      "CandidateStatus": "string",
      "CandidateSteps": [ 
        { 
          "CandidateStepArn": "string",
          "CandidateStepName": "string",
          "CandidateStepType": "string"
        } 
      ],
      "CreationTime": number,
      "EndTime": number,
      "FailureReason": "string",
      "FinalAutoMLJobObjectiveMetric": { 
        "MetricName": "string",
        "Type": "string",
        "Value": number 
      },
      "InferenceContainers": [ 
        { 
          "Environment": { 
            "string": "string"
          },
          "Image": "string"
        }
      ]
    }
  ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Candidates (p. 1107)

Summaries about the Candidates.

Type: Array of AutoMLCandidate (p. 1291) objects

NextToken (p. 1107)

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListCodeRepositories
Service: Amazon SageMaker Service

Gets a list of the Git repositories in your account.

Request Syntax

```
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "LastModifiedTimeAfter": number,
   "LastModifiedTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1109)

A filter that returns only Git repositories that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 1109)

A filter that returns only Git repositories that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 1109)

A filter that returns only Git repositories that were last modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 1109)

A filter that returns only Git repositories that were last modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 1109)

The maximum number of Git repositories to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1109)**

A string in the Git repositories name. This filter returns only repositories whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No

**NextToken (p. 1109)**

If the result of a ListCodeRepositoriesOutput request was truncated, the response includes a NextToken. To get the next set of Git repositories, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**SortBy (p. 1109)**

The field to sort results by. The default is Name.

Type: String

Valid Values: Name | CreationTime | LastModifiedTime

Required: No

**SortOrder (p. 1109)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```json
{
    "CodeRepositorySummaryList": [
        {
            "CodeRepositoryArn": "string",
            "CodeRepositoryName": "string",
            "CreationTime": number,
            "GitConfig": {
                "Branch": "string",
                "RepositoryUrl": "string",
                "SecretArn": "string"
            },
            "LastModifiedTime": number
        }
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CodeRepositorySummaryList (p. 1110)**

Gets a list of summaries of the Git repositories. Each summary specifies the following values for the repository:

- Name
- Amazon Resource Name (ARN)
- Creation time
- Last modified time
- Configuration information, including the URL location of the repository and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.

Type: Array of CodeRepositorySummary (p. 1315) objects

**NextToken (p. 1110)**

If the result of a ListCodeRepositoriesOutput request was truncated, the response includes a NextToken. To get the next set of Git repositories, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListCompilationJobs
Service: Amazon SageMaker Service

Lists model compilation jobs that satisfy various filters.

To create a model compilation job, use CreateCompilationJob (p. 868). To get information about a particular model compilation job you have created, use DescribeCompilationJob (p. 1004).

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1112)

A filter that returns the model compilation jobs that were created after a specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 1112)

A filter that returns the model compilation jobs that were created before a specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 1112)

A filter that returns the model compilation jobs that were modified after a specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 1112)

A filter that returns the model compilation jobs that were modified before a specified time.

Type: Timestamp

Required: No
**MaxResults (p. 1112)**

The maximum number of model compilation jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1112)**

A filter that returns the model compilation jobs whose name contains a specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `[a-zA-Z0-9\-]+`

Required: No

**NextToken (p. 1112)**

If the result of the previous `ListCompilationJobs` request was truncated, the response includes a `NextToken`. To retrieve the next set of model compilation jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `.*`

Required: No

**SortBy (p. 1112)**

The field by which to sort results. The default is `CreationTime`.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 1112)**

The sort order for results. The default is `Ascending`.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1112)**

A filter that retrieves model compilation jobs with a specific `DescribeCompilationJob:CompilationJobStatus` (p. 1005) status.

Type: String

Valid Values: INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED

Required: No
Response Syntax

```json
{
   "CompilationJobSummaries": [
      {
         "CompilationEndTime": number,
         "CompilationJobArn": "string",
         "CompilationJobName": "string",
         "CompilationJobStatus": "string",
         "CompilationStartTime": number,
         "CompilationTargetDevice": "string",
         "CreationTime": number,
         "LastModifiedTime": number
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CompilationJobSummaries (p. 1114)**

An array of CompilationJobSummary (p. 1319) objects, each describing a model compilation job.

Type: Array of CompilationJobSummary (p. 1319) objects

**NextToken (p. 1114)**

If the response is truncated, Amazon SageMaker returns this NextToken. To retrieve the next set of model compilation jobs, use this token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListDomains
Service: Amazon SageMaker Service

Lists the domains.

Request Syntax

```json
{
  "MaxResults": number,
  "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

MaxResults (p. 1116)

Returns a list up to a specified limit.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NextToken (p. 1116)

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

Response Syntax

```json
{
  "Domains": [ 
    {
      "CreationTime": number,
      "DomainArn": "string",
      "DomainId": "string",
      "DomainName": "string",
      "LastModifiedTime": number,
      "Status": "string",
      "Url": "string"
    }
  ],
  "NextToken": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Domains (p. 1116)

The list of domains.

Type: Array of DomainDetails (p. 1339) objects

NextToken (p. 1116)

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListEndpointConfigs
Service: Amazon SageMaker Service

Lists endpoint configurations.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1118)**

A filter that returns only endpoint configurations with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1118)**

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 1118)**

The maximum number of training jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1118)**

A string in the endpoint configuration name. This filter returns only endpoint configurations whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No
NextToken (p. 1118)

If the result of the previous ListEndpointConfig request was truncated, the response includes a NextToken. To retrieve the next set of endpoint configurations, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1118)

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1118)

The sort order for results. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```
{
    "EndpointConfigs": [
        {
            "CreationTime": number,
            "EndpointConfigArn": "string",
            "EndpointConfigName": "string"
        },
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointConfigs (p. 1119)

An array of endpoint configurations.

Type: Array of EndpointConfigSummary (p. 1341) objects

NextToken (p. 1119)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of endpoint configurations, use it in the subsequent request.
Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListEndpoints
Service: Amazon SageMaker Service

Lists endpoints.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1121)**

A filter that returns only endpoints with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1121)**

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1121)**

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1121)**

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

**MaxResults (p. 1121)**

The maximum number of endpoints to return in the response.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 1121)**
A string in endpoint names. This filter returns only endpoints whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-]+
Required: No

**NextToken (p. 1121)**
If the result of a ListEndpoints request was truncated, the response includes a NextToken. To retrieve the next set of endpoints, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No

**SortBy (p. 1121)**
Sorts the list of results. The default is CreationTime.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

**SortOrder (p. 1121)**
The sort order for results. The default is Descending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 1121)**
A filter that returns only endpoints with the specified status.
Type: String
Valid Values: OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed
Required: No

**Response Syntax**

```
"Endpoints": [  
  {  
    "CreationTime": number,
    "EndpointArn": "string",
    "EndpointName": "string",
    "EndpointStatus": "string",
    "LastModifiedTime": number
  },

],

"NextToken": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Endpoints (p. 1122)

An array or endpoint objects.

Type: Array of EndpointSummary (p. 1344) objects

NextToken (p. 1122)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of training jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListExperiments
Service: Amazon SageMaker Service

Lists all the experiments in your account. The list can be filtered to show only experiments that were created in a specific time range. The list can be sorted by experiment name or creation time.

Request Syntax

```
{
  "CreatedAfter": number,
  "CreatedBefore": number,
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreatedAfter (p. 1124)**

A filter that returns only experiments created after the specified time.

Type: Timestamp

Required: No

**CreatedBefore (p. 1124)**

A filter that returns only experiments created before the specified time.

Type: Timestamp

Required: No

**MaxResults (p. 1124)**

The maximum number of experiments to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NextToken (p. 1124)**

If the previous call to ListExperiments didn't return the full set of experiments, the call returns a token for getting the next set of experiments.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *
Required: No

SortBy (p. 1124)

The property used to sort results. The default value is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1124)

The sort order. The default value is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
    "ExperimentSummaries": [
        {
            "CreationTime": number,
            "DisplayName": "string",
            "ExperimentArn": "string",
            "ExperimentName": "string",
            "ExperimentSource": {
                "SourceArn": "string",
                "SourceType": "string"
            },
            "LastModifiedTime": number
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ExperimentSummaries (p. 1125)**

A list of the summaries of your experiments.

Type: Array of ExperimentSummary (p. 1350) objects

**NextToken (p. 1125)**

A token for getting the next set of experiments, if there are any.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListFlowDefinitions
Service: Amazon SageMaker Service

Returns information about the flow definitions in your account.

Request Syntax

```
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "MaxResults": number,
   "NextToken": "string",
   "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1127)**

A filter that returns only flow definitions with a creation time greater than or equal to the specified timestamp.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1127)**

A filter that returns only flow definitions that were created before the specified timestamp.

Type: Timestamp

Required: No

**MaxResults (p. 1127)**

The total number of items to return. If the total number of available items is more than the value specified in MaxResults, then a NextToken will be provided in the output that you can use to resume pagination.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NextToken (p. 1127)**

A token to resume pagination.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No
SortOrder (p. 1127)

An optional value that specifies whether you want the results sorted in Ascending or Descending order.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```
{  "FlowDefinitionSummaries": [  
    {  "CreationTime": number,  
    "FailureReason": "string",  
    "FlowDefinitionArn": "string",  
    "FlowDefinitionName": "string",  
    "FlowDefinitionStatus": "string"  
    },  
    "NextToken": "string"  
  ]}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

FlowDefinitionSummaries (p. 1128)

An array of objects describing the flow definitions.

Type: Array of FlowDefinitionSummary (p. 1360) objects

NextToken (p. 1128)

A token to resume pagination.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
ListHumanTaskUis
Service: Amazon SageMaker Service

Returns information about the human task user interfaces in your account.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NextToken": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1130)**

A filter that returns only human task user interfaces with a creation time greater than or equal to the specified timestamp.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1130)**

A filter that returns only human task user interfaces that were created before the specified timestamp.

Type: Timestamp

Required: No

**MaxResults (p. 1130)**

The total number of items to return. If the total number of available items is more than the value specified in MaxResults, then a NextToken will be provided in the output that you can use to resume pagination.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NextToken (p. 1130)**

A token to resume pagination.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *
**Required:** No

**SortOrder (p. 1130)**

An optional value that specifies whether you want the results sorted in **Ascending** or **Descending** order.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```
{
  "HumanTaskUiSummaries": [
    {
      "CreationTime": number,
      "HumanTaskUiArn": "string",
      "HumanTaskUiName": "string"
    }
  ],
  "NextToken": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**HumanTaskUiSummaries (p. 1131)**

An array of objects describing the human task user interfaces.

Type: Array of HumanTaskUiSummary (p. 1378) objects

**NextToken (p. 1131)**

A token to resume pagination.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
ListHyperParameterTuningJobs
Service: Amazon SageMaker Service

Gets a list of HyperParameterTuningJobSummary (p. 1392) objects that describe the hyperparameter tuning jobs launched in your account.

Request Syntax

```
{
"CreationTimeAfter": number,
"CreationTimeBefore": number,
"LastModifiedTimeAfter": number,
"LastModifiedTimeBefore": number,
"MaxResults": number,
"NameContains": "string",
"NextToken": "string",
"SortBy": "string",
"SortOrder": "string",
"StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1133)**

A filter that returns only tuning jobs that were created after the specified time.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1133)**

A filter that returns only tuning jobs that were created before the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1133)**

A filter that returns only tuning jobs that were modified after the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1133)**

A filter that returns only tuning jobs that were modified before the specified time.

Type: Timestamp

Required: No

**MaxResults (p. 1133)**

The maximum number of tuning jobs to return. The default value is 10.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 1133)**
A string in the tuning job name. This filter returns only tuning jobs whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-\-\~]+
Required: No

**NextToken (p. 1133)**
If the result of the previous ListHyperParameterTuningJobs request was truncated, the response includes a NextToken. To retrieve the next set of tuning jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No

**SortBy (p. 1133)**
The field to sort results by. The default is Name.
Type: String
Valid Values: Name | Status | CreationTime
Required: No

**SortOrder (p. 1133)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 1133)**
A filter that returns only tuning jobs with the specified status.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: No

**Response Syntax**

```json
{
```
"HyperParameterTuningJobSummaries": [
    {
        "CreationTime": number,
        "HyperParameterTuningEndTime": number,
        "HyperParameterTuningJobArn": "string",
        "HyperParameterTuningJobName": "string",
        "HyperParameterTuningJobStatus": "string",
        "LastModifiedTime": number,
        "ObjectiveStatusCounters": {
            "Failed": number,
            "Pending": number,
            "Succeeded": number
        },
        "ResourceLimits": {
            "MaxNumberOfTrainingJobs": number,
            "MaxParallelTrainingJobs": number
        },
        "Strategy": "string",
        "TrainingJobStatusCounters": {
            "Completed": number,
            "InProgress": number,
            "NonRetryableError": number,
            "RetryableError": number,
            "Stopped": number
        }
    }
],
"NextToken": "string"

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

HyperParameterTuningJobSummaries (p. 1134)

A list of HyperParameterTuningJobSummary (p. 1392) objects that describe the tuning jobs that the ListHyperParameterTuningJobs request returned.

Type: Array of HyperParameterTuningJobSummary (p. 1392) objects

NextToken (p. 1134)

If the result of this ListHyperParameterTuningJobs request was truncated, the response includes a NextToken. To retrieve the next set of tuning jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListLabelingJobs
Service: Amazon SageMaker Service

Gets a list of labeling jobs.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1137)**

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1137)**

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1137)**

A filter that returns only labeling jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1137)**

A filter that returns only labeling jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 1137)**

The maximum number of labeling jobs to return in each page of the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 1137)**
A string in the labeling job name. This filter returns only labeling jobs whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: `[a-zA-Z0-9\-]+`
Required: No

**NextToken (p. 1137)**
If the result of the previous ListLabelingJobs request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: `.*`
Required: No

**SortBy (p. 1137)**
The field to sort results by. The default is CreationTime.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

**SortOrder (p. 1137)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 1137)**
A filter that retrieves only labeling jobs with a specific status.
Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: No

**Response Syntax**

```
{
```
"LabelingJobSummaryList": [
  {
    "AnnotationConsolidationLambdaArn": "string",
    "CreationTime": number,
    "FailureReason": "string",
    "InputConfig": {
      "DataAttributes": {
        "ContentClassifiers": [ "string" ]
      },
      "DataSource": {
        "S3DataSource": {
          "ManifestS3Uri": "string"
        }
      }
    },
    "LabelCounters": {
      "FailedNonRetryableError": number,
      "HumanLabeled": number,
      "MachineLabeled": number,
      "TotalLabeled": number,
      "Unlabeled": number
    },
    "LabelingJobArn": "string",
    "LabelingJobName": "string",
    "LabelingJobOutput": {
      "FinalActiveLearningModelArn": "string",
      "OutputDatasetS3Uri": "string"
    },
    "LabelingJobStatus": "string",
    "LastModifiedTime": number,
    "PreHumanTaskLambdaArn": "string",
    "WorkteamArn": "string"
  },
  {
    "NextToken": "string"
  }
],

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobSummaryList (p. 1138)

An array of LabelingJobSummary objects, each describing a labeling job.

Type: Array of LabelingJobSummary (p. 1420) objects

NextToken (p. 1138)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListLabelingJobsForWorkteam
Service: Amazon SageMaker Service

Gets a list of labeling jobs assigned to a specified work team.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "JobReferenceCodeContains": "string",
    "MaxResults": number,
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "WorkteamArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1141)**

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1141)**

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

**JobReferenceCodeContains (p. 1141)**

A filter the limits jobs to only the ones whose job reference code contains the specified string.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: No

**MaxResults (p. 1141)**

The maximum number of labeling jobs to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No
**NextToken (p. 1141)**

If the result of the previous `ListLabelingJobsForWorkteam` request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

- **Type:** String
- **Length Constraints:** Maximum length of 8192.
- **Pattern:** .*
- **Required:** No

**SortBy (p. 1141)**

The field to sort results by. The default is `CreationTime`.

- **Type:** String
- **Valid Values:** `CreationTime`
- **Required:** No

**SortOrder (p. 1141)**

The sort order for results. The default is `Ascending`.

- **Type:** String
- **Valid Values:** `Ascending` | `Descending`
- **Required:** No

**WorkteamArn (p. 1141)**

The Amazon Resource Name (ARN) of the work team for which you want to see labeling jobs for.

- **Type:** String
- **Length Constraints:** Maximum length of 256.
- **Pattern:** `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`
- **Required:** Yes

**Response Syntax**

```json
{
    "LabelingJobSummaryList": [
        {
            "CreationTime": number,
            "JobReferenceCode": "string",
            "LabelCounters": {
                "HumanLabeled": number,
                "PendingHuman": number,
                "Total": number
            },
            "LabelingJobName": "string",
            "NumberOfHumanWorkersPerDataObject": number,
            "WorkRequesterAccountId": "string"
        }
    ],
    "NextToken": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**LabelingJobSummaryList (p. 1142)**

An array of LabelingJobSummary objects, each describing a labeling job.

Type: Array of LabelingJobForWorkteamSummary (p. 1412) objects

**NextToken (p. 1142)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListModelPackages
Service: Amazon SageMaker Service

Lists the model packages that have been created.

Request Syntax

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1144)

A filter that returns only model packages created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 1144)

A filter that returns only model packages created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 1144)

The maximum number of model packages to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 1144)

A string in the model package name. This filter returns only model packages whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-]+

Required: No
NextToken (p. 1144)

If the response to a previous ListModelPackages request was truncated, the response includes a NextToken. To retrieve the next set of model packages, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *

Required: No

SortBy (p. 1144)

The parameter by which to sort the results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1144)

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
    "ModelPackageSummaryList": [
        {
            "CreationTime": number,
            "ModelPackageArn": "string",
            "ModelPackageDescription": "string",
            "ModelPackageName": "string",
            "ModelPackageStatus": "string"
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelPackageSummaryList (p. 1145)

An array of ModelPackageSummary objects, each of which lists a model package.

Type: Array of ModelPackageSummary (p. 1431) objects

NextToken (p. 1145)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of model packages, use it in the subsequent request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**ListModels**  
Service: Amazon SageMaker Service

Lists models created with the CreateModel API.

**Request Syntax**

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters](p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1147)**

A filter that returns only models with a creation time greater than or equal to the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1147)**

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 1147)**

The maximum number of models to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1147)**

A string in the training job name. This filter returns only models in the training job whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No
NextToken (p. 1147)

If the response to a previous ListModels request was truncated, the response includes a NextToken. To retrieve the next set of models, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1147)

Sorts the list of results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1147)

The sort order for results. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
    "Models": [
        {
            "CreationTime": number,
            "ModelArn": "string",
            "ModelName": "string"
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Models (p. 1148)

An array of ModelSummary objects, each of which lists a model.

Type: Array of ModelSummary (p. 1435) objects

NextToken (p. 1148)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of models, use it in the subsequent request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: . *

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListMonitoringExecutions
Service: Amazon SageMaker Service

Returns list of all monitoring job executions.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "EndpointName": "string",
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "MonitoringScheduleName": "string",
    "NextToken": "string",
    "ScheduledTimeAfter": number,
    "ScheduledTimeBefore": number,
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1150)

A filter that returns only jobs created after a specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 1150)

A filter that returns only jobs created before a specified time.

Type: Timestamp

Required: No

EndpointName (p. 1150)

Name of a specific endpoint to fetch jobs for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]*$

Required: No

LastModifiedTimeAfter (p. 1150)

A filter that returns only jobs modified before a specified time.

Type: Timestamp
Required: No

**LastModifiedTimeBefore (p. 1150)**

A filter that returns only jobs modified after a specified time.

Type: Timestamp

Required: No

**MaxResults (p. 1150)**

The maximum number of jobs to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**MonitoringScheduleName (p. 1150)**

Name of a specific schedule to fetch jobs for.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: No

**NextToken (p. 1150)**

The token returned if the response is truncated. To retrieve the next set of job executions, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**ScheduledTimeAfter (p. 1150)**

Filter for jobs scheduled after a specified time.

Type: Timestamp

Required: No

**ScheduledTimeBefore (p. 1150)**

Filter for jobs scheduled before a specified time.

Type: Timestamp

Required: No

**SortBy (p. 1150)**

Whether to sort results by Status, CreationTime, ScheduledTime field. The default is CreationTime.

Type: String
Valid Values: CreationTime | ScheduledTime | Status

Required: No

**SortOrder (p. 1150)**

Whether to sort the results in Ascending or Descending order. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1150)**

A filter that retrieves only jobs with a specific status.

Type: String

Valid Values: Pending | Completed | CompletedWithViolations | InProgress | Failed | Stopping | Stopped

Required: No

**Response Syntax**

```
{
  "MonitoringExecutionSummaries": [
    {
      "CreationTime": number,
      "EndpointName": "string",
      "FailureReason": "string",
      "LastModifiedTime": number,
      "MonitoringExecutionStatus": "string",
      "MonitoringScheduleName": "string",
      "ProcessingJobArn": "string",
      "ScheduledTime": number
    }
  ],
  "NextToken": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**MonitoringExecutionSummaries (p. 1152)**

A JSON array in which each element is a summary for a monitoring execution.

Type: Array of MonitoringExecutionSummary (p. 1442) objects

**NextToken (p. 1152)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.
Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListMonitoringSchedules
Service: Amazon SageMaker Service

Returns list of all monitoring schedules.

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "EndpointName": "string",
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1154)

A filter that returns only monitoring schedules created after a specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 1154)

A filter that returns only monitoring schedules created before a specified time.

Type: Timestamp

Required: No

EndpointName (p. 1154)

Name of a specific endpoint to fetch schedules for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

LastModifiedTimeAfter (p. 1154)

A filter that returns only monitoring schedules modified after a specified time.

Type: Timestamp

Required: No
**LastModifiedTimeBefore (p. 1154)**

A filter that returns only monitoring schedules modified before a specified time.

Type: Timestamp

Required: No

**MaxResults (p. 1154)**

The maximum number of jobs to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1154)**

Filter for monitoring schedules whose name contains a specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-\_]+

Required: No

**NextToken (p. 1154)**

The token returned if the response is truncated. To retrieve the next set of job executions, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**SortBy (p. 1154)**

Whether to sort results by Status, CreationTime, ScheduledTime field. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 1154)**

Whether to sort the results in Ascending or Descending order. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1154)**

A filter that returns only monitoring schedules modified before a specified time.
Type: String

Valid Values: Pending | Failed | Scheduled | Stopped

Required: No

Response Syntax

```json
{
  "MonitoringScheduleSummaries": [
    {
      "CreationTime": number,
      "EndpointName": "string",
      "LastModifiedTime": number,
      "MonitoringScheduleArn": "string",
      "MonitoringScheduleName": "string",
      "MonitoringScheduleStatus": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

MonitoringScheduleSummaries (p. 1156)

A JSON array in which each element is a summary for a monitoring schedule.

Type: Array of MonitoringScheduleSummary (p. 1452) objects

NextToken (p. 1156)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
ListNotebookInstanceLifecycleConfigs
Service: Amazon SageMaker Service

Lists notebook instance lifestyle configurations created with the
CreateNotebookInstanceLifecycleConfig (p. 919) API.

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 1158)

A filter that returns only lifecycle configurations that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 1158)

A filter that returns only lifecycle configurations that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 1158)

A filter that returns only lifecycle configurations that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 1158)

A filter that returns only lifecycle configurations that were modified before the specified time (timestamp).

Type: Timestamp

Required: No
MaxResults (p. 1158)
The maximum number of lifecycle configurations to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

NameContains (p. 1158)
A string in the lifecycle configuration name. This filter returns only lifecycle configurations whose
name contains the specified string.

Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-]+
Required: No

NextToken (p. 1158)
If the result of a ListNotebookInstanceLifecycleConfigs request was truncated, the
response includes a NextToken. To get the next set of lifecycle configurations, use the token in the
next request.

Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No

SortBy (p. 1158)
Sorts the list of results. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime | LastModifiedTime
Required: No

SortOrder (p. 1158)
The sort order for results.

Type: String
Valid Values: Ascending | Descending
Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "NotebookInstanceLifecycleConfigs": [
    {
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1159)

If the response is truncated, Amazon SageMaker returns this token. To get the next set of lifecycle configurations, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

NotebookInstanceLifecycleConfigs (p. 1159)

An array of NotebookInstanceLifecycleConfiguration objects, each listing a lifecycle configuration.

Type: Array of NotebookInstanceLifecycleConfigSummary (p. 1458) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListNotebookInstances
Service: Amazon SageMaker Service

Returns a list of the Amazon SageMaker notebook instances in the requester's account in an AWS Region.

Request Syntax

```json
{
   "AdditionalCodeRepositoryEquals": "string",
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "DefaultCodeRepositoryContains": "string",
   "LastModifiedTimeAfter": number,
   "LastModifiedTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "NotebookInstanceLifecycleConfigNameContains": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**AdditionalCodeRepositoryEquals (p. 1161)**

A filter that returns only notebook instances with associated with the specified git repository.

Type: String


Pattern: ^https://([^/]+)/(.*|[^a-zA-Z0-9]+)(-[a-zA-Z0-9]+)*

Required: No

**CreationTimeAfter (p. 1161)**

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1161)**

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

**DefaultCodeRepositoryContains (p. 1161)**

A string in the name or URL of a Git repository associated with this notebook instance. This filter returns only notebook instances associated with a git repository with a name that contains the specified string.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: [a-zA-Z0-9-]+
Required: No

**LastModifiedTimeAfter (p. 1161)**

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp
Required: No

**LastModifiedTimeBefore (p. 1161)**

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp
Required: No

**MaxResults (p. 1161)**

The maximum number of notebook instances to return.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 1161)**

A string in the notebook instances' name. This filter returns only notebook instances whose name contains the specified string.

Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-]+
Required: No

**NextToken (p. 1161)**

If the previous call to the ListNotebookInstances is truncated, the response includes a NextToken. You can use this token in your subsequent ListNotebookInstances request to fetch the next set of notebook instances.

**Note**

You might specify a filter or a sort order in your request. When response is truncated, you must use the same values for the filter and sort order in the next request.

Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No
**NotebookInstanceLifecycleConfigNameContains (p. 1161)**

A string in the name of a notebook instances lifecycle configuration associated with this notebook instance. This filter returns only notebook instances associated with a lifecycle configuration with a name that contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9][-\[a-zA-Z0-9\]]*$

Required: No

**SortBy (p. 1161)**

The field to sort results by. The default is Name.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 1161)**

The sort order for results.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1161)**

A filter that returns only notebook instances with the specified status.

Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating

Required: No

**Response Syntax**

```json
{
    "NextToken": "string",
    "NotebookInstances": [ 
        {
            "AdditionalCodeRepositories": [ "string" ],
            "CreationTime": number,
            "DefaultCodeRepository": "string",
            "InstanceType": "string",
            "LastModifiedTime": number,
            "NotebookInstanceArn": "string",
            "NotebookInstanceLifecycleConfigName": "string",
            "NotebookInstanceName": "string",
            "NotebookInstanceStatus": "string",
            "Url": "string"
        } 
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1163)

If the response to the previous ListNotebookInstances request was truncated, Amazon SageMaker returns this token. To retrieve the next set of notebook instances, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

NotebookInstances (p. 1163)

An array of NotebookInstanceSummary objects, one for each notebook instance.

Type: Array of NotebookInstanceSummary (p. 1460) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**ListProcessingJobs**  
Service: Amazon SageMaker Service

Lists processing jobs that satisfy various filters.

**Request Syntax**

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1165)**

A filter that returns only processing jobs created after the specified time.

Type: Timestamp  
Required: No

**CreationTimeBefore (p. 1165)**

A filter that returns only processing jobs created after the specified time.

Type: Timestamp  
Required: No

**LastModifiedTimeAfter (p. 1165)**

A filter that returns only processing jobs modified after the specified time.

Type: Timestamp  
Required: No

**LastModifiedTimeBefore (p. 1165)**

A filter that returns only processing jobs modified before the specified time.

Type: Timestamp  
Required: No

**MaxResults (p. 1165)**

The maximum number of processing jobs to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 1165)

A string in the processing job name. This filter returns only processing jobs whose name contains the specified string.

Type: String

Required: No

NextToken (p. 1165)

If the result of the previous ListProcessingJobs request was truncated, the response includes a NextToken. To retrieve the next set of processing jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1165)

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

SortOrder (p. 1165)

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

StatusEquals (p. 1165)

A filter that retrieves only processing jobs with a specific status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

Response Syntax

```json
{
  "NextToken": "string",
  "ProcessingJobSummaries": [
    {
      "CreationTime": number,
      "ExitMessage": "string",
    }
  ]
}
```
"FailureReason": "string",
"LastModifiedTime": number,
"ProcessingEndTime": number,
"ProcessingJobArn": "string",
"ProcessingJobName": "string",
"ProcessingJobStatus": "string"
}
]
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 1166)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of processing jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

**ProcessingJobSummaries (p. 1166)**

An array of `ProcessingJobSummary` objects, each listing a processing job.

Type: Array of `ProcessingJobSummary (p. 1475)` objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListSubscribedWorkteams
Service: Amazon SageMaker Service

Gets a list of the work teams that you are subscribed to in the AWS Marketplace. The list may be empty if no work team satisfies the filter specified in the NameContains parameter.

Request Syntax

```json
{
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**MaxResults (p. 1168)**

The maximum number of work teams to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1168)**

A string in the work team name. This filter returns only work teams whose name contains the specified string.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

**NextToken (p. 1168)**

If the result of the previous ListSubscribedWorkteams request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*  

Required: No

Response Syntax

```json
{
```

1168
"NextToken": "string",
"SubscribedWorkteams": [
  {
    "ListingId": "string",
    "MarketplaceDescription": "string",
    "MarketplaceTitle": "string",
    "SellerName": "string",
    "WorkteamArn": "string"
  }
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1168)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

SubscribedWorkteams (p. 1168)

An array of Workteam objects, each describing a work team.

Type: Array of SubscribedWorkteam (p. 1514) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTags
Service: Amazon SageMaker Service

Returns the tags for the specified Amazon SageMaker resource.

Request Syntax

```json
{
    "MaxResults": number,
    "NextToken": "string",
    "ResourceArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

MaxResults (p. 1170)

- Maximum number of tags to return.
- Type: Integer
- Valid Range: Minimum value of 50.
- Required: No

NextToken (p. 1170)

- If the response to the previous ListTags request is truncated, Amazon SageMaker returns this token. To retrieve the next set of tags, use it in the subsequent request.
- Type: String
- Length Constraints: Maximum length of 8192.
- Pattern: .*
- Required: No

ResourceArn (p. 1170)

- The Amazon Resource Name (ARN) of the resource whose tags you want to retrieve.
- Type: String
- Length Constraints: Maximum length of 256.
- Pattern: arn:.*
- Required: Yes

Response Syntax

```json
{
    "NextToken": "string",
    "Tags": [ ...
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1170)

If response is truncated, Amazon SageMaker includes a token in the response. You can use this token in your subsequent request to fetch next set of tokens.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Tags (p. 1170)

An array of Tag objects, each with a tag key and a value.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrainingJobs
Service: Amazon SageMaker Service

Lists training jobs.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1172)**

A filter that returns only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1172)**

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1172)**

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1172)**

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 1172)**

The maximum number of training jobs to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

NameContains (p. 1172)
A string in the training job name. This filter returns only training jobs whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9\-]+
Required: No

NextToken (p. 1172)
If the result of the previous ListTrainingJobs request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: .*
Required: No

SortBy (p. 1172)
The field to sort results by. The default is CreationTime.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

SortOrder (p. 1172)
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

StatusEquals (p. 1172)
A filter that retrieves only training jobs with a specific status.
Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: No

Response Syntax

```
{
    "NextToken": "string",
```
"TrainingJobSummaries": [
  {
    "CreationTime": number,
    "LastModifiedTime": number,
    "TrainingEndTime": number,
    "TrainingJobArn": "string",
    "TrainingJobName": "string",
    "TrainingJobStatus": "string"
  }
]}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1173)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of
training jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

TrainingJobSummaries (p. 1173)

An array of TrainingJobSummary objects, each listing a training job.

Type: Array of TrainingJobSummary (p. 1531) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrainingJobsForHyperParameterTuningJob

Service: Amazon SageMaker Service

Gets a list of TrainingJobSummary objects that describe the training jobs that a hyperparameter tuning job launched.

Request Syntax

```json
{
   "HyperParameterTuningJobName": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 1175)

The name of the tuning job whose training jobs you want to list.

- Type: String
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`
- Required: Yes

MaxResults (p. 1175)

The maximum number of training jobs to return. The default value is 10.

- Type: Integer
- Valid Range: Minimum value of 1. Maximum value of 100.
- Required: No

NextToken (p. 1175)

If the result of the previous ListTrainingJobsForHyperParameterTuningJob request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.

- Type: String
- Length Constraints: Maximum length of 8192.
- Pattern: `.`
- Required: No

SortBy (p. 1175)

The field to sort results by. The default is Name.
If the value of this field is `FinalObjectiveMetricValue`, any training jobs that did not return an objective metric are not listed.

Type: String

Valid Values: Name | CreationTime | Status | FinalObjectiveMetricValue

Required: No

**SortOrder (p. 1175)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1175)**

A filter that returns only training jobs with the specified status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

**Response Syntax**

```json
{
    "NextToken": "string",
    "TrainingJobSummaries": [
        {
            "CreationTime": number,
            "FailureReason": "string",
            "FinalHyperParameterTuningJobObjectiveMetric": {
                "MetricName": "string",
                "Type": "string",
                "Value": number
            },
            "ObjectiveStatus": "string",
            "TrainingEndTime": number,
            "TrainingJobArn": "string",
            "TrainingJobDefinitionName": "string",
            "TrainingJobName": "string",
            "TrainingJobStatus": "string",
            "TrainingStartTime": number,
            "TunedHyperParameters": {
                "string": "string"
            },
            "TuningJobName": "string"
        }
    ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
NextToken (p. 1176)

If the result of this ListTrainingJobsForHyperParameterTuningJob request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

TrainingJobSummaries (p. 1176)

A list of TrainingJobSummary (p. 1531) objects that describe the training jobs that the ListTrainingJobsForHyperParameterTuningJob request returned.

Type: Array of HyperParameterTrainingJobSummary (p. 1386) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTransformJobs
Service: Amazon SageMaker Service

Lists transform jobs.

Request Syntax

```json
{
"CreationTimeAfter": number,
"CreationTimeBefore": number,
"LastModifiedTimeAfter": number,
"LastModifiedTimeBefore": number,
"MaxResults": number,
"NameContains": "string",
"NextToken": "string",
"SortBy": "string",
"SortOrder": "string",
"StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 1178)**

A filter that returns only transform jobs created after the specified time.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 1178)**

A filter that returns only transform jobs created before the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 1178)**

A filter that returns only transform jobs modified after the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 1178)**

A filter that returns only transform jobs modified before the specified time.

Type: Timestamp

Required: No

**MaxResults (p. 1178)**

The maximum number of transform jobs to return in the response. The default value is 10.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 1178)**

A string in the transform job name. This filter returns only transform jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-]+

Required: No

**NextToken (p. 1178)**

If the result of the previous ListTransformJobs request was truncated, the response includes a NextToken. To retrieve the next set of transform jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**sortBy (p. 1178)**

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**sortOrder (p. 1178)**

The sort order for results. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 1178)**

A filter that retrieves only transform jobs with a specific status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

**Response Syntax**

```json
{
    "NextToken": "string",
```

1179
"TransformJobSummaries": [
    {
        "CreationTime": number,
        "FailureReason": "string",
        "LastModifiedTime": number,
        "TransformEndTime": number,
        "TransformJobArn": "string",
        "TransformJobName": "string",
        "TransformJobStatus": "string"
    }
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 1179)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of transform jobs, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

TransformJobSummaries (p. 1179)

An array of TransformJobSummary objects.

Type: Array of TransformJobSummary (p. 1540) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrialComponents
Service: Amazon SageMaker Service

Lists the trial components in your account. You can sort the list by trial component name or creation time. You can filter the list to show only components that were created in a specific time range. You can also filter on one of the following:

- ExperimentName
- SourceArn
- TrialName

Request Syntax

```
{
  "CreatedAfter": number,
  "CreatedBefore": number,
  "ExperimentName": "string",
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "SourceArn": "string",
  "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CreatedAfter (p. 1181)

A filter that returns only components created after the specified time.

Type: Timestamp

Required: No

CreatedBefore (p. 1181)

A filter that returns only components created before the specified time.

Type: Timestamp

Required: No

ExperimentName (p. 1181)

A filter that returns only components that are part of the specified experiment. If you specify ExperimentName, you can't filter by SourceArn or TrialName.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9-]*

Required: No
MaxResults (p. 1181)

The maximum number of components to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NextToken (p. 1181)

If the previous call to ListTrialComponents didn't return the full set of components, the call returns a token for getting the next set of components.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1181)

The property used to sort results. The default value is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 1181)

The sort order. The default value is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

SourceArn (p. 1181)

A filter that returns only components that have the specified source Amazon Resource Name (ARN). If you specify SourceArn, you can't filter by ExperimentName or TrialName.

Type: String

Length Constraints: Maximum length of 256.

Required: No

TrialName (p. 1181)

A filter that returns only components that are part of the specified trial. If you specify TrialName, you can't filter by ExperimentName or SourceArn.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]*
Required: No

Response Syntax

```
{
    "NextToken": "string",
    "TrialComponentSummaries": [
        {
            "CreatedBy": {
                "DomainId": "string",
                "UserProfileArn": "string",
                "UserProfileName": "string"
            },
            "CreationTime": number,
            "DisplayName": "string",
            "EndTime": number,
            "LastModifiedBy": {
                "DomainId": "string",
                "UserProfileArn": "string",
                "UserProfileName": "string"
            },
            "LastModifiedTime": number,
            "StartTime": number,
            "Status": {
                "Message": "string",
                "PrimaryStatus": "string"
            },
            "TrialComponentArn": "string",
            "TrialComponentName": "string",
            "TrialComponentSource": {
                "SourceArn": "string",
                "SourceType": "string"
            }
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 1183)**

A token for getting the next set of components, if there are any.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

**TrialComponentSummaries (p. 1183)**

A list of the summaries of your trial components.

Type: Array of TrialComponentSummary (p. 1562) objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).
ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrials
Service: Amazon SageMaker Service

Lists the trials in your account. Specify an experiment name to limit the list to the trials that are part of that experiment. The list can be filtered to show only trials that were created in a specific time range. The list can be sorted by trial name or creation time.

Request Syntax

```json
{
   "CreatedAfter": number,
   "CreatedBefore": number,
   "ExperimentName": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CreatedAfter (p. 1185)**

A filter that returns only trials created after the specified time.

Type: Timestamp

Required: No

**CreatedBefore (p. 1185)**

A filter that returns only trials created before the specified time.

Type: Timestamp

Required: No

**ExperimentName (p. 1185)**

A filter that returns only trials that are part of the specified experiment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9][-]*[a-zA-Z0-9-]*$`

Required: No

**MaxResults (p. 1185)**

The maximum number of trials to return in the response. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NextToken (p. 1185)**

If the previous call to `ListTrials` didn't return the full set of trials, the call returns a token for getting the next set of trials.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

**SortBy (p. 1185)**

The property used to sort results. The default value is `CreationTime`.

Type: String

Valid Values: Name | CreationTime

Required: No

**SortOrder (p. 1185)**

The sort order. The default value is `Descending`.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```
{
  "NextToken": "string",
  "TrialSummaries": [
    {
      "CreationTime": number,
      "DisplayName": "string",
      "LastModifiedTime": number,
      "TrialArn": "string",
      "TrialName": "string",
      "TrialSource": {
        "SourceArn": "string",
        "SourceType": "string"
      }
    }
  ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 1186)**

A token for getting the next set of trials, if there are any.
Type: String
Length Constraints: Maximum length of 8192.
Pattern: . *

**TrialSummaries (p. 1186)**
A list of the summaries of your trials.
Type: Array of **TrialSummary (p. 1565)** objects

**Errors**
For information about the errors that are common to all actions, see **Common Errors (p. 1579)**.

**ResourceNotFound**
Resource being access is not found.
HTTP Status Code: 400

**See Also**
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListUserProfiles
Service: Amazon SageMaker Service

Lists user profiles.

Request Syntax

```
{
  "DomainIdEquals": "string",
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "UserProfileNameContains": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

DomainIdEquals (p. 1188)

A parameter by which to filter the results.

Type: String

Length Constraints: Maximum length of 63.

Required: No

MaxResults (p. 1188)

Returns a list up to a specified limit.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NextToken (p. 1188)

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

SortBy (p. 1188)

The parameter by which to sort the results. The default is CreationTime.

Type: String

1188
Valid Values: CreationTime | LastModifiedTime

Required: No

**SortOrder (p. 1188)**

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**UserProfileNameContains (p. 1188)**

A parameter by which to filter the results.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

**Response Syntax**

```json
{
"NextToken": "string",
"UserProfiles": [
{
  "CreationTime": number,
  "DomainId": "string",
  "LastModifiedTime": number,
  "Status": "string",
  "UserProfileName": "string"
}
]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 1189)**

If the previous response was truncated, you will receive this token. Use it in your next request to receive the next set of results.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

**UserProfiles (p. 1189)**

The list of user profiles.

Type: Array of UserProfileDetails (p. 1573) objects
Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListWorkteams
Service: Amazon SageMaker Service

Gets a list of work teams that you have defined in a region. The list may be empty if no work team satisfies the filter specified in the NameContains parameter.

Request Syntax

```
{
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**MaxResults (p. 1191)**

The maximum number of work teams to return in each page of the response.

- Type: Integer
- Valid Range: Minimum value of 1. Maximum value of 100.
- Required: No

**NameContains (p. 1191)**

A string in the work team’s name. This filter returns only work teams whose name contains the specified string.

- Type: String
- Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
- Required: No

**NextToken (p. 1191)**

If the result of the previous ListWorkteams request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

- Type: String
- Length Constraints: Maximum length of 8192.
- Pattern: .*
- Required: No

**SortBy (p. 1191)**

The field to sort results by. The default is CreationTime.
Type: String

Valid Values: Name | CreateDate

Required: No

**SortOrder (p. 1191)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```json
{
  "NextToken": "string",
  "Workteams": [
    {
      "CreateDate": number,
      "Description": "string",
      "LastUpdatedDate": number,
      "MemberDefinitions": [
        {
          "CognitoMemberDefinition": {
            "ClientId": "string",
            "UserGroup": "string",
            "UserPool": "string"
          }
        }
      ],
      "NotificationConfiguration": {
        "NotificationTopicArn": "string"
      },
      "ProductListingIds": [ "string" ],
      "SubDomain": "string",
      "WorkteamArn": "string",
      "WorkteamName": "string"
    }
  ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 1192)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*
Workteams (p. 1192)

An array of Workteam objects, each describing a work team.

Type: Array of Workteam (p. 1578) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
RenderUiTemplate
Service: Amazon SageMaker Service

Renders the UI template so that you can preview the worker's experience.

Request Syntax

```json
{
    "RoleArn": "string",
    "Task": {
        "Input": "string"
    },
    "UiTemplate": {
        "Content": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**RoleArn (p. 1194)**

The Amazon Resource Name (ARN) that has access to the S3 objects that are used by the template.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$

Required: Yes

**Task (p. 1194)**

A RenderableTask object containing a representative task to render.

Type: RenderableTask (p. 1493) object

Required: Yes

**UiTemplate (p. 1194)**

A Template object containing the worker UI template to render.

Type: UiTemplate (p. 1569) object

Required: Yes

Response Syntax

```json
{
    "Errors": [
        {
            "Code": "string",
            "Message": "string"
        }
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Errors (p. 1194)

A list of one or more RenderingError objects if any were encountered while rendering the template. If there were no errors, the list is empty.

Type: Array of RenderingError (p. 1494) objects

RenderedContent (p. 1194)

A Liquid template that renders the HTML for the worker UI.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
Search
Service: Amazon SageMaker Service

Finds Amazon SageMaker resources that match a search query. Matching resource objects are returned as a list of SearchResult objects in the response. You can sort the search results by any resource property in an ascending or descending order.

You can query against the following value types: numeric, text, Boolean, and timestamp.

Request Syntax

```
{
    "MaxResults": number,
    "NextToken": "string",
    "Resource": "string",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "string",
                "Operator": "string",
                "Value": "string"
            }
        ],
        "NestedFilters": [
            {
                "Filters": [
                    {
                        "Name": "string",
                        "Operator": "string",
                        "Value": "string"
                    }
                ],
                "NestedPropertyName": "string"
            }
        ],
        "Operator": "string",
        "SubExpressions": [
            "SearchExpression"
        ]
    },
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

MaxResults (p. 1196)

The maximum number of results to return in a SearchResponse.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No
NextToken (p. 1196)

If more than MaxResults resource objects match the specified SearchExpression, the SearchResponse includes a NextToken. The NextToken can be passed to the next SearchRequest to continue retrieving results for the specified SearchExpression and Sort parameters.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: .*

Required: No

Resource (p. 1196)

The name of the Amazon SageMaker resource to search for.

Type: String

Valid Values: TrainingJob | Experiment | ExperimentTrial | ExperimentTrialComponent

Required: Yes

SearchExpression (p. 1196)

A Boolean conditional statement. Resource objects must satisfy this condition to be included in search results. You must provide at least one subexpression, filter, or nested filter. The maximum number of recursive SubExpressions, NestedFilters, and Filters that can be included in a SearchExpression object is 50.

Type: SearchExpression (p. 1504) object

Required: No

SortBy (p. 1196)

The name of the resource property used to sort the SearchResults. The default is LastModifiedTime.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: No

SortOrder (p. 1196)

How SearchResults are ordered. Valid values are Ascending or Descending. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
}
```
"NextToken": "string",
"Results": [
  {
    "Experiment": {
      "CreatedBy": {
        "DomainId": "string",
        "UserProfileArn": "string",
        "UserProfileName": "string"
      },
      "CreationTime": number,
      "Description": "string",
      "DisplayName": "string",
      "ExperimentArn": "string",
      "ExperimentName": "string",
      "LastModifiedBy": {
        "DomainId": "string",
        "UserProfileArn": "string",
        "UserProfileName": "string"
      },
      "LastModifiedTime": number,
      "Source": {
        "SourceArn": "string",
        "SourceType": "string"
      },
      "Tags": [
        {
          "Key": "string",
          "Value": "string"
        }
      ]
    },
    "TrainingJob": {
      "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "EnableSageMakerMetricsTimeSeries": boolean,
        "MetricDefinitions": [
          {
            "Name": "string",
            "Regex": "string"
          }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
      },
      "AutoMLJobArn": "string",
      "BillableTimeInSeconds": number,
      "CheckpointConfig": {
        "LocalPath": "string",
        "S3Uri": "string"
      },
      "CreationTime": number,
      "DebugHookConfig": {
        "CollectionConfigurations": ["string",
          "CollectionName": "string",
          "CollectionParameters": {
            "string": "string"
          }
        ],
        "HookParameters": {
          "string": "string"
        },
        "LocalPath": "string",
        "S3OutputPath": "string"
      }
    }
  }
]
"DebugRuleConfigurations": [
  {
    "InstanceType": "string",
    "LocalPath": "string",
    "RuleConfigurationName": "string",
    "RuleEvaluatorImage": "string",
    "RuleParameters": {
      "string": "string"
    },
    "S3OutputPath": "string",
    "VolumeSizeInGB": number
  }
],
"DebugRuleEvaluationStatuses": [
  {
    "LastModifiedTime": number,
    "RuleConfigurationName": "string",
    "RuleEvaluationJobArn": "string",
    "RuleEvaluationStatus": "string",
    "StatusDetails": "string"
  }
],
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"ExperimentConfig": {
  "ExperimentName": "string",
  "TrialComponentDisplayName": "string",
  "TrialName": "string"
},
"FailureReason": "string",
"FinalMetricDataList": [
  {
    "MetricName": "string",
    "Timestamp": number,
    "Value": number
  }
],
"HyperParameters": {
  "string": "string"
},
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "FileSystemDataSource": {
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  }
}
"LabelingJobArn": "string",
"LastModifiedTime": number,
"ModelArtifacts": {
  "S3ModelArtifacts": "string"
},
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"SecondaryStatus": "string",
"SecondaryStatusTransitions": [ {
  "EndTime": number,
  "StartTime": number,
  "Status": "string",
  "StatusMessage": "string"
} ],
"StoppingCondition": {
  "MaxRuntimeInSeconds": number,
  "MaxWaitTimeInSeconds": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
],
"TensorBoardOutputConfig": {
  "LocalPath": "string",
  "S3OutputPath": "string"
},
"TrainingEndTime": number,
"TrainingJobArn": "string",
"TrainingJobName": "string",
"TrainingJobStatus": "string",
"TrainingStartTime": number,
"TrainingTimeInSeconds": number,
"TuningJobArn": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
},
"Trial": {
  "CreatedBy": {
    "DomainId": "string",
    "UserProfileArn": "string",
    "UserProfileName": "string"
  },
  "CreationTime": number,
  "DisplayName": "string",
  "ExperimentName": "string",
  "LastModifiedBy": {
    "DomainId": "string",
    "UserProfileArn": "string",
    "UserProfileName": "string"
  }
}
"LastModifiedTime": number,
"Source": {
    "SourceArn": "string",
    "SourceType": "string"
},
"Tags": [
    {
        "Key": "string",
        "Value": "string"
    }
],
"TrialArn": "string",
"TrialComponentSummaries": [
    {
        "CreatedBy": {
            "DomainId": "string",
            "UserProfileArn": "string",
            "UserProfileName": "string"
        },
        "CreationTime": number,
        "TrialComponentArn": "string",
        "TrialComponentName": "string",
        "TrialComponentSource": {
            "SourceArn": "string",
            "SourceType": "string"
        }
    }
],
"TrialName": "string",
"TrialComponent": {
    "CreatedBy": {
        "DomainId": "string",
        "UserProfileArn": "string",
        "UserProfileName": "string"
    },
    "CreationTime": number,
    "DisplayName": "string",
    "EndTime": number,
    "InputArtifacts": {
        "string": {
            "MediaType": "string",
            "Value": "string"
        }
    },
    "LastModifiedBy": {
        "DomainId": "string",
        "UserProfileArn": "string",
        "UserProfileName": "string"
    },
    "LastModifiedTime": number,
    "Metrics": [
        {
            "Avg": number,
            "Count": number,
            "Last": number,
            "Max": number,
            "MetricName": "string",
            "Min": number,
            "SourceArn": "string",
            "StdDev": number,
            "TimeStamp": number
        }
    ],
    "OutputArtifacts": {
        "string": {

        }
    }
}
"MediaType": "string",
"Value": "string"
},
"Parameters": {
"string": {
"NumberValue": number,
"StringValue": "string"
}
},
"Parents": [
{
"ExperimentName": "string",
"TrialName": "string"
}
],
"Source": {
"SourceArn": "string",
"SourceType": "string"
},
"SourceDetail": {
"SourceArn": "string",
"TrainingJob": {
"AlgorithmSpecification": {
"AlgorithmName": "string",
"EnableSageMakerMetricsTimeSeries": boolean,
"MetricDefinitions": [
{
"Name": "string",
"Regex": "string"
}
],
"TrainingImage": "string",
"TrainingInputMode": "string"
},
"AutoMLJobArn": "string",
"BillableTimeInSeconds": number,
"CheckpointConfig": {
"LocalPath": "string",
"S3Uri": "string"
},
"CreationTime": number,
"DebugHookConfig": {
"CollectionConfigurations": [
{
"CollectionName": "string",
"CollectionParameters": {
"string": "string"
}
}
],
"HookParameters": {
"string": "string"
},
"LocalPath": "string",
"S3OutputPath": "string"
},
"DebugRuleConfigurations": [
{
"InstanceType": "string",
"LocalPath": "string",
"RuleConfigurationName": "string",
"RuleEvaluatorImage": "string",
"RuleParameters": {
"string": "string"
}
},
"string": "string"
"S3OutputPath": "string",
"VolumeSizeInGB": number
],
"DebugRuleEvaluationStatuses": [  
  {  
    "LastModifiedTime": number,
    "RuleConfigurationName": "string",
    "RuleEvaluationJobArn": "string",
    "RuleEvaluationStatus": "string",
    "StatusDetails": "string"
  }
],
"EnableInterContainerTrafficEncryption": boolean,
"EnableManagedSpotTraining": boolean,
"EnableNetworkIsolation": boolean,
"ExperimentConfig": {  
  "ExperimentName": "string",
  "TrialComponentDisplayName": "string",
  "TrialName": "string"
},
"FailureReason": "string",
"FinalMetricDataList": [  
  {  
    "MetricName": "string",
    "Timestamp": number,
    "Value": number
  }
],
"HyperParameters": {  
  "string": "string"
},
"InputDataConfig": [  
  {  
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {  
      "FileSystemDataSource": {  
        "DirectoryPath": "string",
        "FileSystemAccessMode": "string",
        "FileSystemId": "string",
        "FileSystemType": "string"
      },
      "S3DataSource": {  
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {  
      "Seed": number
    }
  }
],
"LabelingJobArn": "string",
"LastModifiedTime": number,
"ModelArtifacts": {  
  "S3ModelArtifacts": "string"
},
"OutputDataConfig": {  
  "KmsKeyId": "string",
  "S3OutputPath": "string"
Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**NextToken (p. 1197)**

If the result of the previous `Search` request was truncated, the response includes a `NextToken`. To retrieve the next set of results, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: . *

**Results (p. 1197)**

A list of `SearchResult` objects.

Type: Array of `SearchRecord (p. 1506)` objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StartMonitoringSchedule
Service: Amazon SageMaker Service

Starts a previously stopped monitoring schedule.

Note
New monitoring schedules are immediately started after creation.

Request Syntax

```
{
    "MonitoringScheduleName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

MonitoringScheduleName (p. 1206)

The name of the schedule to start.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StartNotebookInstance
Service: Amazon SageMaker Service

Launches an ML compute instance with the latest version of the libraries and attaches your ML storage volume. After configuring the notebook instance, Amazon SageMaker sets the notebook instance status to InService. A notebook instance's status must be InService before you can connect to your Jupyter notebook.

Request Syntax

```json
{
  "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 1208)**

The name of the notebook instance to start.

- Type: String
- Length Constraints: Maximum length of 63.
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`
- Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StopAutoMLJob
Service: Amazon SageMaker Service

A method for forcing the termination of a running job.

Request Syntax

```
{
    "AutoMLJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AutoMLJobName (p. 1210)

The name of the object you are requesting.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNot_found

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StopCompilationJob
Service: Amazon SageMaker Service

Stops a model compilation job.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal. This gracefully shuts the job down. If the job hasn't stopped, it sends the SIGKILL signal.

When it receives a StopCompilationJob request, Amazon SageMaker changes the CompilationJobSummary:CompilationJobStatus (p. 1319) of the job to Stopping. After Amazon SageMaker stops the job, it sets the CompilationJobSummary:CompilationJobStatus (p. 1319) to Stopped.

Request Syntax

```
{
   "CompilationJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

CompilationJobName (p. 1212)

The name of the model compilation job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StopHyperParameterTuningJob
Service: Amazon SageMaker Service

Stops a running hyperparameter tuning job and all running training jobs that the tuning job launched.

All model artifacts output from the training jobs are stored in Amazon Simple Storage Service (Amazon S3). All data that the training jobs write to Amazon CloudWatch Logs are still available in CloudWatch. After the tuning job moves to the Stopped state, it releases all reserved resources for the tuning job.

Request Syntax

```json
{
   "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 1214)

The name of the tuning job to stop.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]+)*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFoundException

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
### StopLabelingJob

**Service:** Amazon SageMaker Service

Stops a running labeling job. A job that is stopped cannot be restarted. Any results obtained before the job is stopped are placed in the Amazon S3 output bucket.

#### Request Syntax

```json
{
  "LabelingJobName": "string"
}
```

#### Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**LabelingJobName (p. 1216)**

- The name of the labeling job to stop.
- **Type:** String
- **Length Constraints:** Minimum length of 1. Maximum length of 63.
- **Pattern:** \^[a-zA-Z0-9][-]\([\a-zA-Z0-9-]*\)\[^a-zA-Z0-9-]*\]
- **Required:** Yes

#### Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

#### Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

- Resource being access is not found.
- **HTTP Status Code:** 400

#### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StopMonitoringSchedule
Service: Amazon SageMaker Service

Stops a previously started monitoring schedule.

Request Syntax

```json
{
   "MonitoringScheduleName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**MonitoringScheduleName (p. 1218)**

The name of the schedule to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StopNotebookInstance
Service: Amazon SageMaker Service

Terminates the ML compute instance. Before terminating the instance, Amazon SageMaker disconnects the ML storage volume from it. Amazon SageMaker preserves the ML storage volume. Amazon SageMaker stops charging you for the ML compute instance when you call StopNotebookInstance.

To access data on the ML storage volume for a notebook instance that has been terminated, call the StartNotebookInstance API. StartNotebookInstance launches another ML compute instance, configures it, and attaches the preserved ML storage volume so you can continue your work.

Request Syntax

```json
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 1220)**

The name of the notebook instance to terminate.

- **Type:** String
- **Length Constraints:** Maximum length of 63.
- **Pattern:** ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
StopProcessingJob
Service: Amazon SageMaker Service

Stops a processing job.

Request Syntax

```
{
  "ProcessingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**ProcessingJobName (p. 1222)**

The name of the processing job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**StopTrainingJob**  
Service: Amazon SageMaker Service

Stops a training job. To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts, so the results of the training is not lost.

When it receives a `StopTrainingJob` request, Amazon SageMaker changes the status of the job to `Stopping`. After Amazon SageMaker stops the job, it sets the status to `Stopped`.

**Request Syntax**

```json
{
   "TrainingJobName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters](p. 1581).

The request accepts the following data in JSON format.

**TrainingJobName (p. 1224)**

The name of the training job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see [Common Errors](p. 1579).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](p. 1224)
- [AWS SDK for .NET](p. 1224)
- [AWS SDK for C++](p. 1224)
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StopTransformJob
Service: Amazon SageMaker Service

Stops a transform job.

When Amazon SageMaker receives a StopTransformJob request, the status of the job changes to Stopping. After Amazon SageMaker stops the job, the status is set to Stopped. When you stop a transform job before it is completed, Amazon SageMaker doesn't store the job's output in Amazon S3.

Request Syntax

```
{
    "TransformJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

TransformJobName (p. 1226)

  The name of the transform job to stop.

  Type: String


  Pattern: ^[a-zA-20-9][a-zA-20-9]*

  Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceNotFound

  Resource being access is not found.

  HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
UpdateCodeRepository

Service: Amazon SageMaker Service

Updates the specified Git repository with the specified values.

Request Syntax

```json
{
  "CodeRepositoryName": "string",
  "GitConfig": {
    "SecretArn": "string"
  }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 1228)**

The name of the Git repository to update.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**GitConfig (p. 1228)**

The configuration of the git repository, including the URL and the Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the repository. The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```json
{"username": UserName, "password": Password}
```

Type: `GitConfigForUpdate (p. 1363)` object

Required: No

Response Syntax

```json
{
  "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CodeRepositoryArn (p. 1228)**

The ARN of the Git repository.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]\{12\}:code-repository/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateDomain
Service: Amazon SageMaker Service

Updates a domain. Changes will impact all of the people in the domain.

Request Syntax

```
{
  "DefaultUserSettings": {
    "ExecutionRole": "string",
    "JupyterServerAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "KernelGatewayAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    },
    "SecurityGroups": [ "string" ],
    "SharingSettings": {
      "NotebookOutputOption": "string",
      "S3KmsKeyId": "string",
      "S3OutputPath": "string"
    },
    "TensorBoardAppSettings": {
      "DefaultResourceSpec": {
        "EnvironmentArn": "string",
        "InstanceType": "string"
      }
    }
  },
  "DomainId": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DefaultUserSettings (p. 1230)**

A collection of settings.

Type: UserSettings (p. 1575) object

Required: No

**DomainId (p. 1230)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes
Response Syntax

```
{
    "DomainArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**DomainArn (p. 1231)**

The domain Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:domain/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateEndpoint
Service: Amazon SageMaker Service

Deploys the new EndpointConfig specified in the request, switches to using newly created endpoint, and then deletes resources provisioned for the endpoint using the previous EndpointConfig (there is no availability loss).

When Amazon SageMaker receives the request, it sets the endpoint status to Updating. After updating the endpoint, it sets the status to InService. To check the status of an endpoint, use the DescribeEndpoint API.

**Note**
You must not delete an EndpointConfig in use by an endpoint that is live or while the UpdateEndpoint or CreateEndpoint operations are being performed on the endpoint. To update an endpoint, you must create a new EndpointConfig.

**Request Syntax**

```
{
  "EndpointConfigName": "string",
  "EndpointName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 1233)**

The name of the new endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**EndpointName (p. 1233)**

The name of the endpoint whose configuration you want to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**Response Syntax**

```
{
  "EndpointArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**EndpointArn (p. 1233)**

The Amazon Resource Name (ARN) of the endpoint.

- **Type:** String
- **Length Constraints:** Minimum length of 20. Maximum length of 2048.
- **Pattern:** \`arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*\`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

**HTTP Status Code:** 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateEndpointWeightsAndCapacities
Service: Amazon SageMaker Service

Updates variant weight of one or more variants associated with an existing endpoint, or capacity of one variant associated with an existing endpoint. When it receives the request, Amazon SageMaker sets the endpoint status to Updating. After updating the endpoint, it sets the status to InService. To check the status of an endpoint, use the DescribeEndpoint API.

Request Syntax

```
{
  "DesiredWeightsAndCapacities": [
    {
      "DesiredInstanceCount": number,
      "DesiredWeight": number,
      "VariantName": "string"
    }
  ],
  "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DesiredWeightsAndCapacities (p. 1235)**

An object that provides new capacity and weight values for a variant.

Type: Array of DesiredWeightAndCapacity (p. 1338) objects

Array Members: Minimum number of 1 item.

Required: Yes

**EndpointName (p. 1235)**

The name of an existing Amazon SageMaker endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

Response Syntax

```
{
  "EndpointArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**EndpointArn** *(p. 1235)*

The Amazon Resource Name (ARN) of the updated endpoint.

- **Type:** String
- **Length Constraints:** Minimum length of 20. Maximum length of 2048.
- **Pattern:** `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*`

**Errors**

For information about the errors that are common to all actions, see [Common Errors](p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

- **HTTP Status Code:** 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateExperiment
Service: Amazon SageMaker Service

Adds, updates, or removes the description of an experiment. Updates the display name of an experiment.

Request Syntax

```json
{
    "Description": "string",
    "DisplayName": "string",
    "ExperimentName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**Description (p. 1237)**

The description of the experiment.

Type: String

Length Constraints: Maximum length of 3072.

Pattern: .*

Required: No

**DisplayName (p. 1237)**

The name of the experiment as displayed. The name doesn't need to be unique. If DisplayName isn't specified, ExperimentName is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*    

Required: No

**ExperimentName (p. 1237)**

The name of the experiment to update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*    

Required: Yes

Response Syntax

```json
{
}
```
"ExperimentArn": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ExperimentArn (p. 1237)**

The Amazon Resource Name (ARN) of the experiment.

Type: String

Length Constraints: Maximum length of 256.

Pattern: \( \text{arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment/.*} \)

Errors

For information about the errors that are common to all actions, see [Common Errors (p. 1579)].

**ConflictException**

There was a conflict when you attempted to modify an experiment, trial, or trial component.

HTTP Status Code: 400

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)
UpdateMonitoringSchedule
Service: Amazon SageMaker Service

Updates a previously created schedule.

Request Syntax

```json
{
  "MonitoringScheduleConfig": {
    "MonitoringJobDefinition": {
      "BaselineConfig": {
        "ConstraintsResource": {
          "S3Uri": "string"
        },
        "StatisticsResource": {
          "S3Uri": "string"
        }
      },
      "Environment": {
        "string": "string"
      },
      "MonitoringAppSpecification": {
        "ContainerArguments": [ "string" ],
        "ContainerEntrypoint": [ "string" ],
        "ImageUri": "string",
        "PostAnalyticsProcessorSourceUri": "string",
        "RecordPreprocessorSourceUri": "string"
      },
      "MonitoringInputs": [
        {
          "EndpointInput": {
            "EndpointName": "string",
            "LocalPath": "string",
            "S3DataDistributionType": "string",
            "S3InputMode": "string"
          }
        }
      ],
      "MonitoringOutputConfig": {
        "KmsKeyId": "string",
        "MonitoringOutputs": [
          {
            "S3Output": {
              "LocalPath": "string",
              "S3UploadMode": "string",
              "S3Uri": "string"
            }
          }
        ],
        "MonitoringResources": {
          "ClusterConfig": {
            "InstanceCount": number,
            "InstanceType": "string",
            "VolumeKmsKeyId": "string",
            "VolumeSizeInGB": number
          }
        },
        "NetworkConfig": {
          "EnableNetworkIsolation": boolean,
          "VpcConfig": {
            "SecurityGroupIds": [ "string" ],
            "Subnets": [ "string" ]
          }
        }
      }
    }
  }
}
```


### Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**MonitoringScheduleConfig (p. 1239)**

The configuration object that specifies the monitoring schedule and defines the monitoring job.

Type: MonitoringScheduleConfig (p. 1451) object

Required: Yes

**MonitoringScheduleName (p. 1239)**

The name of the monitoring schedule. The name must be unique within an AWS Region within an AWS account.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

### Response Syntax

```
{
  "MonitoringScheduleArn": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**MonitoringScheduleArn (p. 1240)**

The Amazon Resource Name (ARN) of the monitoring schedule.

Type: String

Length Constraints: Maximum length of 256.
Pattern: . *

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateNotebookInstance
Service: Amazon SageMaker Service

Updates a notebook instance. NotebookInstance updates include upgrading or downgrading the ML compute instance used for your notebook instance to accommodate changes in your workload requirements.

Request Syntax

```
{
   "AcceleratorTypes": [ "string" ],
   "AdditionalCodeRepositories": [ "string" ],
   "DefaultCodeRepository": "string",
   "DisassociateAcceleratorTypes": boolean,
   "DisassociateAdditionalCodeRepositories": boolean,
   "DisassociateDefaultCodeRepository": boolean,
   "DisassociateLifecycleConfig": boolean,
   "InstanceType": "string",
   "LifecycleConfigName": "string",
   "NotebookInstanceName": "string",
   "RoleArn": "string",
   "RootAccess": "string",
   "VolumeSizeInGB": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

AcceleratorTypes (p. 1242)

A list of the Elastic Inference (EI) instance types to associate with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge | ml.eia2.medium | ml.eia2.large | ml.eia2.xlarge

Required: No

AdditionalCodeRepositories (p. 1242)

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://([^/]+)/(.*$|$^[a-zA-Z0-9]+(-*[a-zA-Z0-9]*)
Required: No

**DefaultCodeRepository (p. 1242)**

The Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see [Associating Git Repositories with Amazon SageMaker Notebook Instances](#).

Type: String


Pattern: `^https://([^/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**DisassociateAcceleratorTypes (p. 1242)**

A list of the Elastic Inference (EI) instance types to remove from this notebook instance. This operation is idempotent. If you specify an accelerator type that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

**DisassociateAdditionalCodeRepositories (p. 1242)**

A list of names or URLs of the default Git repositories to remove from this notebook instance. This operation is idempotent. If you specify a Git repository that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

**DisassociateDefaultCodeRepository (p. 1242)**

The name or URL of the default Git repository to remove from this notebook instance. This operation is idempotent. If you specify a Git repository that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

**DisassociateLifecycleConfig (p. 1242)**

Set to `true` to remove the notebook instance lifecycle configuration currently associated with the notebook instance. This operation is idempotent. If you specify a lifecycle configuration that is not associated with the notebook instance when you call this method, it does not throw an error.

Type: Boolean

Required: No

**InstanceType (p. 1242)**

The Amazon ML compute instance type.

Type: String

Valid Values: `ml.t2.medium` | `ml.t2.large` | `ml.t2.xlarge` | `ml.t2.2xlarge` | `ml.t3.medium` | `ml.t3.large` | `ml.t3.xlarge` | `ml.t3.2xlarge` | `ml.m4.xlarge`
| ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge |
| ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge |
| ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge |
| ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
| ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge |
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge |

Required: No

**LifecycleConfigName (p. 1242)**

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

**NotebookInstanceName (p. 1242)**

The name of the notebook instance to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**RoleArn (p. 1242)**

The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access the notebook instance. For more information, see Amazon SageMaker Roles.

Note

To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-zA-Z\-]*:iam::\d{12}:role/?[a-zA-Z0-9=+.@\-_/]+$?

Required: No

**RootAccess (p. 1242)**

Whether root access is enabled or disabled for users of the notebook instance. The default value is Enabled.

Note

If you set this to Disabled, users don't have root access on the notebook instance, but lifecycle configuration scripts still run with root permissions.

Type: String

Valid Values: Enabled | Disabled
Required: No

**VolumeSizeInGB (p. 1242)**

The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB. ML storage volumes are encrypted, so Amazon SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

Type: Integer


Required: No

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**UpdateNotebookInstanceLifecycleConfig**

Service: Amazon SageMaker Service

Updates a notebook instance lifecycle configuration created with the `CreateNotebookInstanceLifecycleConfig (p. 919)` API.

**Request Syntax**

```json
{
    "NotebookInstanceLifecycleConfigName": "string",
    "OnCreate": [
        {
            "Content": "string"
        }
    ],
    "OnStart": [
        {
            "Content": "string"
        }
    ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 1246)**

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  

Required: Yes

**OnCreate (p. 1246)**

The shell script that runs only once, when you create a notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects

Array Members: Maximum number of 1 item.

Required: No

**OnStart (p. 1246)**

The shell script that runs every time you start a notebook instance, including when you create the notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 1459) objects

Array Members: Maximum number of 1 item.

Required: No

1246
Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateTrial
Service: Amazon SageMaker Service

Updates the display name of a trial.

Request Syntax

```
{
  "DisplayName": "string",
  "TrialName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DisplayName (p. 1248)**

The name of the trial as displayed. The name doesn't need to be unique. If **DisplayName** isn't specified, **TrialName** is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)`

Required: No

**TrialName (p. 1248)**

The name of the trial to update.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)`

Required: Yes

Response Syntax

```
{
  "TrialArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialArn (p. 1248)**

The Amazon Resource Name (ARN) of the trial.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial/.*

Errors
For information about the errors that are common to all actions, see Common Errors (p. 1579).

ConflictException
There was a conflict when you attempted to modify an experiment, trial, or trial component.
HTTP Status Code: 400

ResourceNotFoundException
Resource being access is not found.
HTTP Status Code: 400

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateTrialComponent
Service: Amazon SageMaker Service

Updates one or more properties of a trial component.

Request Syntax

```json
{
    "DisplayName": "string",
    "EndTime": number,
    "InputArtifacts": {
        "string": {
            "MediaType": "string",
            "Value": "string"
        }
    },
    "InputArtifactsToRemove": [ "string" ],
    "OutputArtifacts": {
        "string": {
            "MediaType": "string",
            "Value": "string"
        }
    },
    "OutputArtifactsToRemove": [ "string" ],
    "Parameters": {
        "string": {
            "NumberValue": number,
            "StringValue": "string"
        }
    },
    "ParametersToRemove": [ "string" ],
    "StartTime": number,
    "Status": {
        "Message": "string",
        "PrimaryStatus": "string"
    },
    "TrialComponentName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DisplayName (p. 1250)**

The name of the component as displayed. The name doesn't need to be unique. If DisplayName isn't specified, TrialComponentName is displayed.

- **Type:** String
- **Length Constraints:** Minimum length of 1. Maximum length of 82.
- **Pattern:** ^[a-zA-Z0-9](\-*[a-zA-Z0-9])*  
- **Required:** No

**EndTime (p. 1250)**

When the component ended.
Type: Timestamp
Required: No

**InputArtifacts (p. 1250)**

Replaces all of the component's input artifacts with the specified artifacts.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: . *

Required: No

**InputArtifactsToRemove (p. 1250)**

The input artifacts to remove from the component.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**OutputArtifacts (p. 1250)**

Replaces all of the component's output artifacts with the specified artifacts.

Type: String to `TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: . *

Required: No

**OutputArtifactsToRemove (p. 1250)**

The output artifacts to remove from the component.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**Parameters (p. 1250)**

Replaces all of the component's hyperparameters with the specified hyperparameters.

Type: String to `TrialComponentParameterValue (p. 1556)` object map

Key Length Constraints: Maximum length of 256.

Key Pattern: . *

Required: No

**ParametersToRemove (p. 1250)**

The hyperparameters to remove from the component.
Type: Array of strings
Length Constraints: Maximum length of 256.
Pattern: .*
Required: No
**StartTime (p. 1250)**
When the component started.
Type: Timestamp
Required: No
**Status (p. 1250)**
The new status of the component.
Type: `TrialComponentStatus (p. 1561)` object
Required: No
**TrialComponentName (p. 1250)**
The name of the component to update.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])+*
Required: Yes

### Response Syntax

```json
{
    "TrialComponentArn": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrialComponentArn (p. 1252)**
The Amazon Resource Name (ARN) of the trial component.
Type: String
Length Constraints: Maximum length of 256.
Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*`

### Errors

For information about the errors that are common to all actions, see [Common Errors (p. 1579)](#).
ConflictException

There was a conflict when you attempted to modify an experiment, trial, or trial component.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateUserProfile
Service: Amazon SageMaker Service

Updates a user profile.

Request Syntax

```json
{
   "DomainId": "string",
   "UserProfileName": "string",
   "UserSettings": {
      "ExecutionRole": "string",
      "JupyterServerAppSettings": {
         "DefaultResourceSpec": {
            "EnvironmentArn": "string",
            "InstanceType": "string"
         }
      },
      "KernelGatewayAppSettings": {
         "DefaultResourceSpec": {
            "EnvironmentArn": "string",
            "InstanceType": "string"
         }
      },
      "SecurityGroups": [ "string" ],
      "SharingSettings": {
         "NotebookOutputOption": "string",
         "S3KmsKeyId": "string",
         "S3OutputPath": "string"
      },
      "TensorBoardAppSettings": {
         "DefaultResourceSpec": {
            "EnvironmentArn": "string",
            "InstanceType": "string"
         }
      }
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

**DomainId (p. 1254)**

The domain ID.

Type: String

Length Constraints: Maximum length of 63.

Required: Yes

**UserProfileName (p. 1254)**

The user profile name.

Type: String

Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

UserSettings (p. 1254)

A collection of settings.

Type: UserSettings (p. 1575) object

Required: No

Response Syntax

```json
{
   "UserProfileArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

UserProfileArn (p. 1255)

The user profile Amazon Resource Name (ARN).

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:user-profile/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
UpdateWorkteam
Service: Amazon SageMaker Service

Updates an existing work team with new member definitions or description.

Request Syntax

```
{
    "Description": "string",
    "MemberDefinitions": [
        {
            "CognitoMemberDefinition": {
                "ClientId": "string",
                "UserGroup": "string",
                "UserPool": "string"
            }
        }
    ],
    "NotificationConfiguration": {
        "NotificationTopicArn": "string"
    }
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 1581).

The request accepts the following data in JSON format.

Description (p. 1257)

An updated description for the work team.

Type: String


Pattern: .+

Required: No

MemberDefinitions (p. 1257)

A list of MemberDefinition objects that contain the updated work team members.

Type: Array of MemberDefinition (p. 1423) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: No

NotificationConfiguration (p. 1257)

Configures SNS topic notifications for available or expiring work items

Type: NotificationConfiguration (p. 1463) object

Required: No
WorkteamName (p. 1257)

The name of the work team to update.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Syntax

```
{  
  "Workteam": {  
    "CreateDate": number,  
    "Description": "string",  
    "LastUpdatedDate": number,  
    "MemberDefinitions": [  
      {  
        "CognitoMemberDefinition": {  
          "ClientId": "string",  
          "UserGroup": "string",  
          "UserPool": "string"  
        }  
      },  
    ]  
  },  
  "NotificationConfiguration": {  
    "NotificationTopicArn": "string"  
  },  
  "ProductListingIds": [ "string" ],  
  "SubDomain": "string",  
  "WorkteamArn": "string",  
  "WorkteamName": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Workteam (p. 1258)**

A `Workteam` object that describes the updated work team.

Type: `Workteam (p. 1578)` object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

**ResourceLimitExceededException**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Amazon SageMaker Runtime

The following actions are supported by Amazon SageMaker Runtime:

- InvokeEndpoint (p. 1260)
InvokeEndpoint
Service: Amazon SageMaker Runtime

After you deploy a model into production using Amazon SageMaker hosting services, your client applications use this API to get inferences from the model hosted at the specified endpoint.

For an overview of Amazon SageMaker, see How It Works.

Amazon SageMaker strips all POST headers except those supported by the API. Amazon SageMaker might add additional headers. You should not rely on the behavior of headers outside those enumerated in the request syntax.

Calls to InvokeEndpoint are authenticated by using AWS Signature Version 4. For information, see Authenticating Requests (AWS Signature Version 4) in the Amazon S3 API Reference.

A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

**Note**
Endpoints are scoped to an individual account, and are not public. The URL does not contain the account ID, but Amazon SageMaker determines the account ID from the authentication token that is supplied by the caller.

**Request Syntax**

```plaintext
POST /endpoints/EndpointName/invocations HTTP/1.1
Content-Type: ContentType
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes
X-Amzn-SageMaker-Target-Model: TargetModel

Body
```

**URI Request Parameters**

The request requires the following URI parameters.

**Accept (p. 1260)**

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

**ContentType (p. 1260)**

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

**CustomAttributes (p. 1260)**

Provides additional information about a request for an inference submitted to a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to provide an ID that you can use to track a request or to provide other metadata that a service endpoint was programmed to process. The value must consist of no
more than 1024 visible US-ASCII characters as specified in Section 3.3.6. Field Value Components of the Hypertext Transfer Protocol (HTTP/1.1). This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

**EndpointName (p. 1260)**

The name of the endpoint that you specified when you created the endpoint using the CreateEndpoint API.

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* TargetModel (p. 1260)

The model to request for inference when invoking a multi-model endpoint.


Pattern: \A\S[\p{Print}]\z

**Request Body**

The request accepts the following binary data.

**Body (p. 1260)**

Provides input data, in the format specified in the Content-Type request header. Amazon SageMaker passes all of the data in the body to the model.

For information about the format of the request body, see Common Data Formats—Inference.

Length Constraints: Maximum length of 5242880.

**Response Syntax**

HTTP/1.1 200
Content-Type: Content-Type
x-Amzn-Invoked-Production-Variant: InvokedProductionVariant
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes

**Body**

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

**ContentType (p. 1261)**

The MIME type of the inference returned in the response body.

Length Constraints: Maximum length of 1024.
CustomAttributes (p. 1261)

Provides additional information in the response about the inference returned by a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to return an ID received in the CustomAttributes header of a request or other metadata that a service endpoint was programmed to produce. The value must consist of no more than 1024 visible US-ASCII characters as specified in Section 3.3.6. Field Value Components of the Hypertext Transfer Protocol (HTTP/1.1). If the customer wants the custom attribute returned, the model must set the custom attribute to be included on the way back.

This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.

InvokedProductionVariant (p. 1261)

Identifies the production variant that was invoked.

Length Constraints: Maximum length of 1024.

The response returns the following as the HTTP body.

Body (p. 1261)

Includes the inference provided by the model.

For information about the format of the response body, see Common Data Formats—Inference.

Length Constraints: Maximum length of 5242880.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 1579).

InternalFailure

An internal failure occurred.

HTTP Status Code: 500

ModelError

Model (owned by the customer in the container) returned 4xx or 5xx error code.

HTTP Status Code: 424

ServiceUnavailable

The service is unavailable. Try your call again.

HTTP Status Code: 503

ValidationError

Inspect your request and try again.
HTTP Status Code: 400

Example

Pass a trace ID in the CustomAttribute of a request and return it in the CustomAttribute of the response.

In this example a trace ID is passed to the service endpoint in the CustomAttributes header of the request and then retrieved and returned in the CustomAttributes header of the response.

Sample Request

```python
import boto3
custom_attributes = "c000b4f9-df62-4c85-a0bf-7c525f9104a4"  # An example of a trace ID.
endpoint_name = "..."  # Your endpoint name.
content_type = "..."  # The MIME type of the input data in the request body.
accept = "..."  # The desired MIME type of the inference in the response.
payload = "..."  # Payload for inference.

client = boto3.client('sagemaker-runtime')
response = client.invoke_endpoint(
    EndpointName=endpoint_name,
   CustomAttributes=custom_attributes,
   ContentType=content_type,
   Accept=accept,
   Body=payload
)

print(response['CustomAttributes'])  # If model receives and updates the custom_attributes header
# by adding "Trace id: " in front of custom_attributes in the request,
becomes
# "Trace ID: c000b4f9-
df62-4c85-a0bf-7c525f9104a4"
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
Data Types

The following data types are supported by Amazon SageMaker Service:

- AlgorithmSpecification (p. 1274)
- AlgorithmStatusDetails (p. 1276)
- AlgorithmStatusItem (p. 1277)
- AlgorithmSummary (p. 1278)
- AlgorithmValidationProfile (p. 1280)
- AlgorithmValidationSpecification (p. 1281)
- AnnotationConsolidationConfig (p. 1282)
- AppDetails (p. 1288)
- AppSpecification (p. 1290)
- AutoMLCandidate (p. 1291)
- AutoMLCandidateStep (p. 1293)
- AutoMLChannel (p. 1294)
- AutoMLContainerDefinition (p. 1295)
- AutoMLDataSource (p. 1296)
- AutoMLJobArtifacts (p. 1297)
- AutoMLJobCompletionCriteria (p. 1298)
- AutoMLJobConfig (p. 1299)
- AutoMLJobObjective (p. 1300)
- AutoMLJobSummary (p. 1301)
- AutoMLOutputDataConfig (p. 1303)
- AutoMLS3DataSource (p. 1304)
- AutoMLSecurityConfig (p. 1305)
- CaptureContentTypeHeader (p. 1306)
- CaptureOption (p. 1307)
- CategoricalParameterRange (p. 1308)
- CategoricalParameterRangeSpecification (p. 1309)
- Channel (p. 1310)
- ChannelSpecification (p. 1312)
- CheckpointConfig (p. 1314)
- CodeRepositorySummary (p. 1315)
- CognitoMemberDefinition (p. 1317)
- CollectionConfiguration (p. 1318)
- CompilationJobSummary (p. 1319)
- ContainerDefinition (p. 1321)
- ContinuousParameterRange (p. 1323)
- ContinuousParameterRangeSpecification (p. 1325)
- DataCaptureConfig (p. 1326)
- DataCaptureConfigSummary (p. 1327)
- DataProcessing (p. 1328)
Data Types

- DataSource (p. 1330)
- DebugHookConfig (p. 1331)
- DebugRuleConfiguration (p. 1333)
- DebugRuleEvaluationStatus (p. 1335)
- DeployedImage (p. 1337)
- DesiredWeightAndCapacity (p. 1338)
- DomainDetails (p. 1339)
- EndpointConfigSummary (p. 1341)
- EndpointInput (p. 1342)
- EndpointSummary (p. 1344)
- Experiment (p. 1346)
- ExperimentConfig (p. 1348)
- ExperimentSource (p. 1349)
- ExperimentSummary (p. 1350)
- FileSystemDataSource (p. 1352)
- Filter (p. 1354)
- FinalAutoMLJobObjectiveMetric (p. 1357)
- FinalHyperParameterTuningJobObjectiveMetric (p. 1358)
- FlowDefinitionOutputConfig (p. 1359)
- FlowDefinitionSummary (p. 1360)
- GitConfig (p. 1362)
- GitConfigForUpdate (p. 1363)
- HumanLoopActivationConditionsConfig (p. 1364)
- HumanLoopActivationConfig (p. 1365)
- HumanLoopConfig (p. 1366)
- HumanLoopRequestSource (p. 1371)
- HumanTaskConfig (p. 1372)
- HumanTaskUiSummary (p. 1378)
- HyperParameterAlgorithmSpecification (p. 1379)
- HyperParameterSpecification (p. 1381)
- HyperParameterTrainingJobDefinition (p. 1383)
- HyperParameterTrainingJobSummary (p. 1386)
- HyperParameterTuningJobConfig (p. 1389)
- HyperParameterTuningJobObjective (p. 1391)
- HyperParameterTuningJobSummary (p. 1392)
- HyperParameterTuningJobWarmStartConfig (p. 1394)
- InferenceSpecification (p. 1396)
- InputConfig (p. 1398)
- IntegerParameterRange (p. 1400)
- IntegerParameterRangeSpecification (p. 1402)
- JupyterServerAppSettings (p. 1403)
- KernelGatewayAppSettings (p. 1404)
- LabelCounters (p. 1405)
- LabelCountersForWorkteam (p. 1407)
- LabelingJobAlgorithmsConfig (p. 1408)
- LabelingJobDataAttributes (p. 1410)
Data Types

- LabelingJobDataSource (p. 1411)
- LabelingJobForWorkteamSummary (p. 1412)
- LabelingJobInputConfig (p. 1414)
- LabelingJobOutput (p. 1415)
- LabelingJobOutputConfig (p. 1416)
- LabelingJobResourceConfig (p. 1417)
- LabelingJobS3DataSource (p. 1418)
- LabelingJobStoppingConditions (p. 1419)
- LabelingJobSummary (p. 1420)
- MemberDefinition (p. 1423)
- MetricData (p. 1424)
- MetricDefinition (p. 1425)
- ModelArtifacts (p. 1426)
- ModelPackageContainerDefinition (p. 1427)
- ModelPackageStatusDetails (p. 1429)
- ModelPackageStatusItem (p. 1430)
- ModelPackageSummary (p. 1431)
- ModelPackageValidationProfile (p. 1433)
- ModelPackageValidationSpecification (p. 1434)
- ModelSummary (p. 1435)
- MonitoringAppSpecification (p. 1436)
- MonitoringBaselineConfig (p. 1438)
- MonitoringClusterConfig (p. 1439)
- MonitoringConstraintsResource (p. 1441)
- MonitoringExecutionSummary (p. 1442)
- MonitoringInput (p. 1444)
- MonitoringJobDefinition (p. 1445)
- MonitoringOutput (p. 1447)
- MonitoringOutputConfig (p. 1448)
- MonitoringResources (p. 1449)
- MonitoringS3Output (p. 1450)
- MonitoringScheduleConfig (p. 1451)
- MonitoringScheduleSummary (p. 1452)
- MonitoringStatisticsResource (p. 1454)
- MonitoringStoppingCondition (p. 1455)
- NestedFilters (p. 1456)
- NetworkConfig (p. 1457)
- NotebookInstanceLifecycleConfigSummary (p. 1458)
- NotebookInstanceLifecycleHook (p. 1459)
- NotebookInstanceSummary (p. 1460)
- NotificationConfiguration (p. 1463)
- ObjectiveStatusCounters (p. 1464)
- OutputConfig (p. 1465)
- OutputDataConfig (p. 1466)
- ParameterRange (p. 1468)
- ParameterRanges (p. 1469)
• Parent (p. 1470)
• ParentHyperParameterTuningJob (p. 1471)
• ProcessingClusterConfig (p. 1472)
• ProcessingInput (p. 1474)
• ProcessingJobSummary (p. 1475)
• ProcessingOutput (p. 1477)
• ProcessingOutputConfig (p. 1478)
• ProcessingResources (p. 1479)
• ProcessingS3Input (p. 1480)
• ProcessingS3Output (p. 1482)
• ProcessingStoppingCondition (p. 1483)
• ProductionVariant (p. 1484)
• ProductionVariantSummary (p. 1486)
• PropertyNameQuery (p. 1488)
• PropertyNameSuggestion (p. 1489)
• PublicWorkforceTaskPrice (p. 1490)
• RenderableTask (p. 1493)
• RenderingError (p. 1494)
• ResolvedAttributes (p. 1495)
• ResourceConfig (p. 1496)
• ResourceLimits (p. 1498)
• ResourceSpec (p. 1499)
• RetentionPolicy (p. 1500)
• S3DataSource (p. 1501)
• ScheduleConfig (p. 1503)
• SearchExpression (p. 1504)
• SearchRecord (p. 1506)
• SecondaryStatusTransition (p. 1507)
• SharingSettings (p. 1509)
• ShuffleConfig (p. 1510)
• SourceAlgorithm (p. 1511)
• SourceAlgorithmSpecification (p. 1512)
• StoppingCondition (p. 1513)
• SubscribedWorkteam (p. 1514)
• SuggestionQuery (p. 1516)
• Tag (p. 1517)
• TensorBoardAppSettings (p. 1518)
• TensorBoardOutputConfig (p. 1519)
• TrainingJob (p. 1520)
• TrainingJobDefinition (p. 1527)
• TrainingJobStatusCounters (p. 1529)
• TrainingJobSummary (p. 1531)
• TrainingSpecification (p. 1533)
• TransformDataSource (p. 1535)
• TransformInput (p. 1536)
• TransformJobDefinition (p. 1538)
The following data types are supported by Amazon SageMaker Runtime:

**Amazon SageMaker Service**

The following data types are supported by Amazon SageMaker Service:

- AlgorithmSpecification (p. 1274)
- AlgorithmStatusDetails (p. 1276)
- AlgorithmStatusItem (p. 1277)
- AlgorithmSummary (p. 1278)
- AlgorithmValidationProfile (p. 1280)
- AlgorithmValidationSpecification (p. 1281)
- AnnotationConsolidationConfig (p. 1282)
- AppDetails (p. 1288)
- AppSpecification (p. 1290)
- AutoMLCandidate (p. 1291)
- AutoMLCandidateStep (p. 1293)
- AutoMLChannel (p. 1294)
- AutoMLContainerDefinition (p. 1295)
- AutoMLDataSource (p. 1296)
- AutoMLJobArtifacts (p. 1297)
- AutoMLJobCompletionCriteria (p. 1298)
- AutoMLJobConfig (p. 1299)
- AutoMLJobObjective (p. 1300)
- AutoMLJobSummary (p. 1301)
- AutoMLOutputDataConfig (p. 1303)
- AutoMLS3DataSource (p. 1304)
- AutoMLSecurityConfig (p. 1305)
- CaptureContentTypeHeader (p. 1306)
- CaptureOption (p. 1307)
- CategoricalParameterRange (p. 1308)
- CategoricalParameterRangeSpecification (p. 1309)
- Channel (p. 1310)
- ChannelSpecification (p. 1312)
- CheckpointConfig (p. 1314)
- CodeRepositorySummary (p. 1315)
- CognitoMemberDefinition (p. 1317)
- CollectionConfiguration (p. 1318)
- CompilationJobSummary (p. 1319)
- ContainerDefinition (p. 1321)
- ContinuousParameterRange (p. 1323)
- ContinuousParameterRangeSpecification (p. 1325)
- DataCaptureConfig (p. 1326)
- DataCaptureConfigSummary (p. 1327)
- DataProcessing (p. 1328)
- DataSource (p. 1330)
- DebugHookConfig (p. 1331)
- DebugRuleConfiguration (p. 1333)
- DebugRuleEvaluationStatus (p. 1335)
- DeployedImage (p. 1337)
- DesiredWeightAndCapacity (p. 1338)
- DomainDetails (p. 1339)
- EndpointConfigSummary (p. 1341)
- EndpointInput (p. 1342)
- EndpointSummary (p. 1344)
- Experiment (p. 1346)
- ExperimentConfig (p. 1348)
- ExperimentSource (p. 1349)
- ExperimentSummary (p. 1350)
- FileSystemDataSource (p. 1352)
- Filter (p. 1354)
- FinalAutoMLJobObjectiveMetric (p. 1357)
- FinalHyperParameterTuningJobObjectiveMetric (p. 1358)
- FlowDefinitionOutputConfig (p. 1359)
- FlowDefinitionSummary (p. 1360)
• GitConfig (p. 1362)
• GitConfigForUpdate (p. 1363)
• HumanLoopActivationConditionsConfig (p. 1364)
• HumanLoopActivationConfig (p. 1365)
• HumanLoopConfig (p. 1366)
• HumanLoopRequestSource (p. 1371)
• HumanTaskConfig (p. 1372)
• HumanTaskUiSummary (p. 1378)
• HyperParameterAlgorithmSpecification (p. 1379)
• HyperParameterSpecification (p. 1381)
• HyperParameterTrainingJobDefinition (p. 1383)
• HyperParameterTrainingJobSummary (p. 1386)
• HyperParameterTuningJobConfig (p. 1389)
• HyperParameterTuningJobObjective (p. 1391)
• HyperParameterTuningJobSummary (p. 1392)
• HyperParameterTuningJobWarmStartConfig (p. 1394)
• InferenceSpecification (p. 1396)
• InputConfig (p. 1398)
• IntegerParameterRange (p. 1400)
• IntegerParameterRangeSpecification (p. 1402)
• JupyterServerAppSettings (p. 1403)
• KernelGatewayAppSettings (p. 1404)
• LabelCounters (p. 1405)
• LabelCountersForWorkteam (p. 1407)
• LabelingJobAlgorithmsConfig (p. 1408)
• LabelingJobDataAttributes (p. 1410)
• LabelingJobDataSource (p. 1411)
• LabelingJobForWorkteamSummary (p. 1412)
• LabelingJobInputConfig (p. 1414)
• LabelingJobOutput (p. 1415)
• LabelingJobOutputConfig (p. 1416)
• LabelingJobResourceConfig (p. 1417)
• LabelingJobS3DataSource (p. 1418)
• LabelingJobStoppingConditions (p. 1419)
• LabelingJobSummary (p. 1420)
• MemberDefinition (p. 1423)
• MetricData (p. 1424)
• MetricDefinition (p. 1425)
• ModelArtifacts (p. 1426)
• ModelPackageContainerDefinition (p. 1427)
• ModelPackageStatusDetails (p. 1429)
• ModelPackageStatusItem (p. 1430)
• ModelPackageSummary (p. 1431)
• ModelPackageValidationProfile (p. 1433)
• ModelPackageValidationSpecification (p. 1434)
• ModelSummary (p. 1435)
• MonitoringAppSpecification (p. 1436)
• MonitoringBaselineConfig (p. 1438)
• MonitoringClusterConfig (p. 1439)
• MonitoringConstraintsResource (p. 1441)
• MonitoringExecutionSummary (p. 1442)
• MonitoringInput (p. 1444)
• MonitoringJobDefinition (p. 1445)
• MonitoringOutput (p. 1447)
• MonitoringOutputConfig (p. 1448)
• MonitoringResources (p. 1449)
• MonitoringS3Output (p. 1450)
• MonitoringScheduleConfig (p. 1451)
• MonitoringScheduleSummary (p. 1452)
• MonitoringStatisticsResource (p. 1454)
• MonitoringStoppingCondition (p. 1455)
• NestedFilters (p. 1456)
• NetworkConfig (p. 1457)
• NotebookInstanceLifecycleConfigSummary (p. 1458)
• NotebookInstanceLifecycleHook (p. 1459)
• NotebookInstanceSummary (p. 1460)
• NotificationConfiguration (p. 1463)
• ObjectiveStatusCounters (p. 1464)
• OutputConfig (p. 1465)
• OutputDataConfig (p. 1466)
• ParameterRange (p. 1468)
• ParameterRanges (p. 1469)
• Parent (p. 1470)
• ParentHyperParameterTuningJob (p. 1471)
• ProcessingClusterConfig (p. 1472)
• ProcessingInput (p. 1474)
• ProcessingJobSummary (p. 1475)
• ProcessingOutput (p. 1477)
• ProcessingOutputConfig (p. 1478)
• ProcessingResources (p. 1479)
• ProcessingS3Input (p. 1480)
• ProcessingS3Output (p. 1482)
• ProcessingStoppingCondition (p. 1483)
• ProductionVariant (p. 1484)
• ProductionVariantSummary (p. 1486)
• PropertyNameQuery (p. 1488)
• PropertyNameSuggestion (p. 1489)
• PublicWorkforceTaskPrice (p. 1490)
• RenderableTask (p. 1493)
• RenderingError (p. 1494)
• ResolvedAttributes (p. 1495)
• ResourceConfig (p. 1496)
• ResourceLimits (p. 1498)
• ResourceSpec (p. 1499)
• RetentionPolicy (p. 1500)
• S3DataSource (p. 1501)
• ScheduleConfig (p. 1503)
• SearchExpression (p. 1504)
• SearchRecord (p. 1506)
• SecondaryStatusTransition (p. 1507)
• SharingSettings (p. 1509)
• ShuffleConfig (p. 1510)
• SourceAlgorithm (p. 1511)
• SourceAlgorithmSpecification (p. 1512)
• StoppingCondition (p. 1513)
• SubscribedWorkteam (p. 1514)
• SuggestionQuery (p. 1516)
• Tag (p. 1517)
• TensorBoardAppSettings (p. 1518)
• TensorBoardOutputConfig (p. 1519)
• TrainingJob (p. 1520)
• TrainingJobDefinition (p. 1527)
• TrainingJobStatusCounters (p. 1529)
• TrainingJobSummary (p. 1531)
• TrainingSpecification (p. 1533)
• TransformDataSource (p. 1535)
• TransformInput (p. 1536)
• TransformJobDefinition (p. 1538)
• TransformJobSummary (p. 1540)
• TransformOutput (p. 1542)
• TransformResources (p. 1544)
• TransformS3DataSource (p. 1546)
• Trial (p. 1548)
• TrialComponent (p. 1550)
• TrialComponentArtifact (p. 1553)
• TrialComponentMetricSummary (p. 1554)
• TrialComponentParameterValue (p. 1556)
• TrialComponentSimpleSummary (p. 1557)
• TrialComponentSource (p. 1559)
• TrialComponentSourceDetail (p. 1560)
• TrialComponentStatus (p. 1561)
• TrialComponentSummary (p. 1562)
• TrialSource (p. 1564)
• TrialSummary (p. 1565)
• TuningJobCompletionCriteria (p. 1567)
• UICfg (p. 1568)
• UIConfig (p. 1569)
• UI_TemplateInfo (p. 1570)
• USD (p. 1571)
• UserContext (p. 1572)
• UserProfileDetails (p. 1573)
• UserSettings (p. 1575)
• VpcConfig (p. 1577)
• Workteam (p. 1578)
AlgorithmSpecification
Service: Amazon SageMaker Service

Specifies the training algorithm to use in a CreateTrainingJob request.

For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about using your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Contents

AlgorithmName

The name of the algorithm resource to use for the training job. This must be an algorithm resource that you created or subscribe to on AWS Marketplace. If you specify a value for this parameter, you can't specify a value for TrainingImage.

Type: String
Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\/)??([a-zA- zA-20-9]*[a-zA-20-9]*[0-62]1)?<^->-#
Required: No

EnableSageMakerMetricsTimeSeries

To generate and save time-series metrics during training, set to true. The default is false and time-series metrics aren't generated except in the following cases:
- You use one of the Amazon SageMaker built-in algorithms
- You use one of the following Prebuilt Amazon SageMaker Docker Images:
  - Tensorflow (version >= 1.15)
  - MXNet (version >= 1.6)
  - PyTorch (version >= 1.3)
- You specify at least one MetricDefinition (p. 1425)

Type: Boolean
Required: No

MetricDefinitions

A list of metric definition objects. Each object specifies the metric name and regular expressions used to parse algorithm logs. Amazon SageMaker publishes each metric to Amazon CloudWatch.

Type: Array of MetricDefinition (p. 1425) objects
Array Members: Minimum number of 0 items. Maximum number of 40 items.
Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters. Amazon SageMaker supports both registry/repository[:tag] and registry/repository[@digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String
Length Constraints: Maximum length of 255.

Pattern: .*

Required: No

**TrainingInputMode**

The input mode that the algorithm supports. For the input modes that Amazon SageMaker algorithms support, see [Algorithms](https://aws.amazon.com/sagemaker/). If an algorithm supports the **File** input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the **Pipe** input mode, Amazon SageMaker streams data directly from S3 to the container.

In **File** mode, make sure you provision ML storage volume with sufficient capacity to accommodate the data download from S3. In addition to the training data, the ML storage volume also stores the output model. The algorithm container use ML storage volume to also store intermediate information, if any.

For distributed algorithms using **File** mode, training data is distributed uniformly, and your training duration is predictable if the input data objects size is approximately same. Amazon SageMaker does not split the files any further for model training. If the object sizes are skewed, training won't be optimal as the data distribution is also skewed where one host in a training cluster is overloaded, thus becoming bottleneck in training.

Type: String

Valid Values: **Pipe** | **File**

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmStatusDetails
Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the algorithm.

Contents

ImageScanStatuses
The status of the scan of the algorithm's Docker image container.
Type: Array of AlgorithmStatusItem (p. 1277) objects
Required: No

ValidationStatuses
The status of algorithm validation.
Type: Array of AlgorithmStatusItem (p. 1277) objects
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmStatusItem
Service: Amazon SageMaker Service

Represents the overall status of an algorithm.

Contents

FailureReason

if the overall status is Failed, the reason for the failure.

Type: String
Required: No

Name

The name of the algorithm for which the overall status is being reported.

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]+)*$

Required: Yes

Status

The current status.

Type: String

Valid Values: NotStarted | InProgress | Completed | Failed

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmSummary
Service: Amazon SageMaker Service

Provides summary information about an algorithm.

Contents

AlgorithmArn
The Amazon Resource Name (ARN) of the algorithm.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:algorithm/.*
Required: Yes

AlgorithmDescription
A brief description of the algorithm.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*
Required: No

AlgorithmName
The name of the algorithm that is described by the summary.
Type: String
Pattern: ^[a-zA-Z0-9-]*[^a-zA-Z0-9-]*/*
Required: Yes

AlgorithmStatus
The overall status of the algorithm.
Type: String
Valid Values: Pending | InProgress | Completed | Failed | Deleting
Required: Yes

CreationTime
A timestamp that shows when the algorithm was created.
Type: Timestamp
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
AlgorithmValidationProfile

Service: Amazon SageMaker Service

Defines a training job and a batch transform job that Amazon SageMaker runs to validate your algorithm.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

Contents

**ProfileName**

The name of the profile for the algorithm. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

**TrainingJobDefinition**

The TrainingJobDefinition object that describes the training job that Amazon SageMaker runs to validate your algorithm.

Type: TrainingJobDefinition (p. 1527) object

Required: Yes

**TransformJobDefinition**

The TransformJobDefinition object that describes the transform job that Amazon SageMaker runs to validate your algorithm.

Type: TransformJobDefinition (p. 1538) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmValidationSpecification
Service: Amazon SageMaker Service

Specifies configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Contents

ValidationProfiles

An array of AlgorithmValidationProfile objects, each of which specifies a training job and batch transform job that Amazon SageMaker runs to validate your algorithm.

Type: Array of AlgorithmValidationProfile objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole

The IAM roles that Amazon SageMaker uses to run the training jobs.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@+-_/]+$  

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**AnnotationConsolidationConfig**

Service: Amazon SageMaker Service

Configures how labels are consolidated across human workers.

**Contents**

**AnnotationConsolidationLambdaArn**

The Amazon Resource Name (ARN) of a Lambda function implements the logic for annotation consolidation.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

- **Bounding box** - Finds the most similar boxes from different workers based on the Jaccard index of the boxes.

  - arn:aws:lambda:us-east-1:432418664414:function:ACS-BoundingBox
  - arn:aws:lambda:eu-west-1:568282634449:function:ACS-BoundingBox
  - arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-BoundingBox
  - arn:aws:lambda:ap-south-1:565803892007:function:ACS-BoundingBox
  - arn:aws:lambda:eu-central-1:203001061592:function:ACS-BoundingBox
  - arn:aws:lambda:eu-west-2:487402164563:function:ACS-BoundingBox
  - arn:aws:lambda:ap-southeast-1:377565633583:function:ACS-BoundingBox
  - arn:aws:lambda:ca-central-1:918755190332:function:ACS-BoundingBox

- **Image classification** - Uses a variant of the Expectation Maximization approach to estimate the true class of an image based on annotations from individual workers.

  - arn:aws:lambda:us-east-1:432418664414:function:ACS-ImageMultiClass
  - arn:aws:lambda:eu-west-1:568282634449:function:ACS-ImageMultiClass
**Semantic segmentation** - Treats each pixel in an image as a multi-class classification and treats pixel annotations from workers as "votes" for the correct label.

**Text classification** - Uses a variant of the Expectation Maximization approach to estimate the true class of text based on annotations from individual workers.
• **Named entity recognition** - Groups similar selections and calculates aggregate boundaries, resolving to most-assigned label.


• **Bounding box verification** - Uses a variant of the Expectation Maximization approach to estimate the true class of verification judgement for bounding box labels based on annotations from individual workers.

  arn:aws:lambda:us-east-1:432418664414:function:ACS-VerificationBoundingBox
  arn:aws:lambda:us-east-2:266458841044:function:ACS-VerificationBoundingBox
  arn:aws:lambda:us-west-2:081040173940:function:ACS-VerificationBoundingBox
  arn:aws:lambda:eu-west-2:568282634449:function:ACS-VerificationBoundingBox
  arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-VerificationBoundingBox
  arn:aws:lambda:ap-south-1:565803892007:function:ACS-VerificationBoundingBox
  arn:aws:lambda:ap-northeast-2:845288260483:function:ACS-VerificationBoundingBox
• *Semantic segmentation verification* - Uses a variant of the Expectation Maximization approach to estimate the true class of verification judgement for semantic segmentation labels based on annotations from individual workers.

• *Bounding box adjustment* - Finds the most similar boxes from different workers based on the Jaccard index of the adjusted annotations.
• **Semantic segmentation adjustment** - Treats each pixel in an image as a multi-class classification and treats pixel adjusted annotations from workers as "votes" for the correct label.

For more information, see Annotation Consolidation.
Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d(12):function:[a-zA-Z0-9-_.]+(:($LATEST|[a-zA-Z0-9-_.]+))?

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AppDetails
Service: Amazon SageMaker Service

The app's details.

Contents

AppName
The name of the app.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
Required: No

AppType
The type of app.
Type: String
Valid Values: JupyterServer | KernelGateway | TensorBoard
Required: No

CreationTime
The creation time.
Type: Timestamp
Required: No

DomainId
The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: No

Status
The status.
Type: String
Valid Values: Deleted | Deleting | Failed | InService | Pending
Required: No

UserProfileName
The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^\[a-zA-Z0-9](-*[a-zA-Z0-9])*\n
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**AppSpecification**

Service: Amazon SageMaker Service

Configuration to run a processing job in a specified container image.

**Contents**

**ContainerArguments**

The arguments for a container used to run a processing job.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 100 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**ContainerEntrypoint**

The entrypoint for a container used to run a processing job.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 100 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**ImageUri**

The container image to be run by the processing job.

Type: String

Length Constraints: Maximum length of 255.

Pattern: . *

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLCandidate
Service: Amazon SageMaker Service

An AutoPilot job will return recommendations, or candidates. Each candidate has further details about
the steps involved, and the status.

Contents

CandidateName
The candidate name.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Required: Yes

CandidateStatus
The candidate's status.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: Yes

CandidateSteps
The candidate's steps.
Type: Array of AutoMLCandidateStep (p. 1293) objects
Required: Yes

CreationTime
The creation time.
Type: Timestamp
Required: Yes

EndTime
The end time.
Type: Timestamp
Required: No

FailureReason
The failure reason.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

FinalAutoMLJobObjectiveMetric
The candidate result from a job.
Type: FinalAutoMLJobObjectiveMetric (p. 1357) object

Required: No

InferenceContainers

The inference containers.

Type: Array of AutoMLContainerDefinition (p. 1295) objects

Array Members: Maximum number of 5 items.

Required: No

LastModifiedTime

The last modified time.

Type: Timestamp

Required: Yes

ObjectiveStatus

The objective status.

Type: String

Valid Values: Succeeded | Pending | Failed

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLCandidateStep

Service: Amazon SageMaker Service

Information about the steps for a Candidate, and what step it is working on.

Contents

CandidateStepArn

The ARN for the Candidate's step.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:.*

Required: Yes

CandidateStepName

The name for the Candidate's step.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Required: Yes

CandidateStepType

Whether the Candidate is at the transform, training, or processing step.

Type: String

Valid Values: AWS::SageMaker::TrainingJob | AWS::SageMaker::TransformJob | AWS::SageMaker::ProcessingJob

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLChannel
Service: Amazon SageMaker Service

Similar to Channel. A channel is a named input source that training algorithms can consume. Refer to Channel for detailed descriptions.

Contents

CompressionType

You can use Gzip or None. The default value is None.

Type: String

Valid Values: None | Gzip

Required: No

DataSource

The data source.

Type: AutoMLDataSource (p. 1296) object

Required: Yes

TargetAttributeName

The name of the target variable in supervised learning, a.k.a. ‘y’.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLContainerDefinition
Service: Amazon SageMaker Service

A list of container definitions that describe the different containers that make up one AutoML candidate. Refer to ContainerDefinition for more details.

Contents

Environment

Environment variables to set in the container. Refer to ContainerDefinition for more details.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: [a-zA-Z\_][a-zA-Z0-9\_]*

Value Length Constraints: Maximum length of 1024.

Value Pattern: [\s\s]*

Required: No

Image

The ECR path of the container. Refer to ContainerDefinition for more details.

Type: String

Length Constraints: Maximum length of 255.

Pattern: [\S]+

Required: Yes

ModelDataUrl

The location of the model artifacts. Refer to ContainerDefinition for more details.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLDataSource
Service: Amazon SageMaker Service

The data source for the AutoPilot job.

Contents

S3DataSource

The Amazon S3 location of the data.

Type: AutoMLS3DataSource (p. 1304) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLJobArtifacts

Service: Amazon SageMaker Service

Artifacts that are generated during a job.

Contents

CandidateDefinitionNotebookLocation

  The URL to the notebook location.

  Type: String

  Length Constraints: Minimum length of 1.

  Required: No

DataExplorationNotebookLocation

  The URL to the notebook location.

  Type: String

  Length Constraints: Minimum length of 1.

  Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLJobCompletionCriteria
Service: Amazon SageMaker Service

How long a job is allowed to run, or how many candidates a job is allowed to generate.

Contents

MaxAutoMLJobRuntimeInSeconds
The maximum time, in seconds, an AutoML job is allowed to wait for a trial to complete. It must be equal to or greater than MaxRuntimePerTrainingJobInSeconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

MaxCandidates
The maximum number of times a training job is allowed to run.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

MaxRuntimePerTrainingJobInSeconds
The maximum time, in seconds, a job is allowed to run.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLJobConfig

Service: Amazon SageMaker Service

A collection of settings used for a job.

Contents

CompletionCriteria

How long a job is allowed to run, or how many candidates a job is allowed to generate.

Type: AutoMLJobCompletionCriteria (p. 1298) object

Required: No

SecurityConfig

Security configuration for traffic encryption or Amazon VPC settings.

Type: AutoMLSecurityConfig (p. 1305) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLJobObjective
Service: Amazon SageMaker Service

Applies a metric to minimize or maximize for the job’s objective.

Contents

MetricName

The name of the metric.

Type: String

Valid Values: Accuracy | MSE | F1 | F1macro

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLJobSummary
Service: Amazon SageMaker Service

Provides a summary about a job.

Contents

AutoMLJobArn
The ARN of the job.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:automl-job/.*
Required: Yes

AutoMLJobName
The name of the object you are requesting.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*
Required: Yes

AutoMLJobSecondaryStatus
The job’s secondary status.
Type: String
Valid Values: Starting | AnalyzingData | FeatureEngineering | ModelTuning | MaxCandidatesReached | Failed | Stopped | MaxAutoMLJobRuntimeReached | Stopping | CandidateDefinitionsGenerated
Required: Yes

AutoMLJobStatus
The job’s status.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: Yes

CreationTime
When the job was created.
Type: Timestamp
Required: Yes

EndTime
The end time.
Type: Timestamp
Required: No

**FailureReason**

The failure reason.

Type: String
Length Constraints: Maximum length of 1024.
Required: No

**LastModifiedTime**

When the job was last modified.

Type: Timestamp
Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLOutputDataConfig
Service: Amazon SageMaker Service

The output data configuration.

Contents

KmsKeyId
The AWS KMS encryption key ID.
Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: No

S3OutputPath
The Amazon S3 output path. Must be 128 characters or less.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3):/([^/]+)//([^/]+)?(.*)$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLS3DataSource
Service: Amazon SageMaker Service

The Amazon S3 data source.

Contents

S3DataType

The data type.

Type: String

Valid Values: ManifestFile | S3Prefix

Required: Yes

S3Uri

The URL to the Amazon S3 data source.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/?(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AutoMLSecurityConfig
Service: Amazon SageMaker Service

Security options.

Contents

EnableInterContainerTrafficEncryption

Whether to use traffic encryption between the container layers.

Type: Boolean
Required: No

VolumeKmsKeyId

The key used to encrypt stored data.

Type: String
Length Constraints: Maximum length of 2048.
Pattern: . *
Required: No

VpcConfig

VPC configuration.

Type: VpcConfig (p. 1577) object
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CaptureContentTypeHeader
Service: Amazon SageMaker Service

Contents

CsvContentTypes
Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 10 items.
Length Constraints: Minimum length of 1.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*\/[a-zA-Z0-9](-*[a-zA-Z0-9.])*  
Required: No

JsonContentTypes
Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 10 items.
Length Constraints: Minimum length of 1.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*\/[a-zA-Z0-9](-*[a-zA-Z0-9.])*  
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
CaptureOption
Service: Amazon SageMaker Service

Contents

CaptureMode
  Type: String
  Valid Values: Input | Output
  Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**CategoricalParameterRange**
Service: Amazon SageMaker Service

A list of categorical hyperparameters to tune.

**Contents**

**Name**

The name of the categorical hyperparameter to tune.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

**Values**

A list of the categories for the hyperparameter.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CategoricalParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for a categorical hyperparameter.

Contents

Values

The allowed categories for the hyperparameter.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**Channel**

Service: Amazon SageMaker Service

A channel is a named input source that training algorithms can consume.

**Contents**

**ChannelName**

The name of the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: [A-Za-z0-9\._-]+

Required: Yes

**CompressionType**

If training data is compressed, the compression type. The default value is None. CompressionType is used only in Pipe input mode. In File mode, leave this field unset or set it to None.

Type: String

Valid Values: None | Gzip

Required: No

**ContentType**

The MIME type of the data.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: No

**DataSource**

The location of the channel data.

Type: DataSource (p. 1330) object

Required: Yes

**InputMode**

(Optional) The input mode to use for the data channel in a training job. If you don't set a value for InputMode, Amazon SageMaker uses the value set for TrainingInputMode. Use this parameter to override the TrainingInputMode setting in a AlgorithmSpecification (p. 1274) request when you have a channel that needs a different input mode from the training job's general setting. To download the data from Amazon Simple Storage Service (Amazon S3) to the provisioned ML storage volume, and mount the directory to a Docker volume, use File input mode. To stream data directly from Amazon S3 to the container, choose Pipe input mode.

To use a model for incremental training, choose File input model.

Type: String
Valid Values: Pipe | File
Required: No

RecordWrapperType

Specify RecordIO as the value when input data is in raw format but the training algorithm requires the RecordIO format. In this case, Amazon SageMaker wraps each individual S3 object in a RecordIO record. If the input data is already in RecordIO format, you don't need to set this attribute. For more information, see Create a Dataset Using RecordIO.

In File mode, leave this field unset or set it to None.

Type: String
Valid Values: None | RecordIO
Required: No

ShuffleConfig

A configuration for a shuffle option for input data in a channel. If you use S3Prefix for S3DataType, this shuffles the results of the $3 key prefix matches. If you use ManifestFile, the order of the S3 object references in the ManifestFile is shuffled. If you use AugmentedManifestFile, the order of the JSON lines in the AugmentedManifestFile is shuffled. The shuffling order is determined using the Seed value.

For Pipe input mode, shuffling is done at the start of every epoch. With large datasets this ensures that the order of the training data is different for each epoch, it helps reduce bias and possible overfitting. In a multi-node training job when ShuffleConfig is combined with S3DataDistributionType of ShardedByS3Key, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

Type: ShuffleConfig (p. 1510) object
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ChannelSpecification**

Service: Amazon SageMaker Service

Defines a named input source, called a channel, to be used by an algorithm.

**Contents**

**Description**

A brief description of the channel.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

**IsRequired**

Indicates whether the channel is required by the algorithm.

Type: Boolean

Required: No

**Name**

The name of the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: [A-Za-z0-9.\-_]+*

Required: Yes

**SupportedCompressionTypes**

The allowed compression types, if data compression is used.

Type: Array of strings

Valid Values: None | Gzip

Required: No

**SupportedContentTypes**

The supported MIME types for the data.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

**SupportedInputModes**

The allowed input mode, either FILE or PIPE.
In FILE mode, Amazon SageMaker copies the data from the input source onto the local Amazon Elastic Block Store (Amazon EBS) volumes before starting your training algorithm. This is the most commonly used input mode.

In PIPE mode, Amazon SageMaker streams input data from the source directly to your algorithm without using the EBS volume.

Type: Array of strings

Array Members: Minimum number of 1 item.

Valid Values: Pipe | File

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**CheckpointConfig**

Service: Amazon SageMaker Service

Contains information about the output location for managed spot training checkpoint data.

**Contents**

**LocalPath**

(Optional) The local directory where checkpoints are written. The default directory is `/opt/ml/checkpoints/`.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: .*

Required: No

**S3Uri**

Identifies the S3 path where you want Amazon SageMaker to store checkpoints. For example, `s3://bucket-name/key-name-prefix`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^([https|s3]:/(/[^/]+)/?([^/]*$`)

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**CodeRepositorySummary**  
Service: Amazon SageMaker Service

Specifies summary information about a Git repository.

**Contents**

**CodeRepositoryArn**  
The Amazon Resource Name (ARN) of the Git repository.  
Type: String  
Length Constraints: Minimum length of 1. Maximum length of 2048.  
Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]\{12\}:code-repository/.*`  
Required: Yes

**CodeRepositoryName**  
The name of the Git repository.  
Type: String  
Pattern: `^[a-zA-Z0-9\-\]{0,63}$`  
Required: Yes

**CreationTime**  
The date and time that the Git repository was created.  
Type: Timestamp  
Required: Yes

**GitConfig**  
Configuration details for the Git repository, including the URL where it is located and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.  
Type: `GitConfig (p. 1362)` object  
Required: No

**LastModifiedTime**  
The date and time that the Git repository was last modified.  
Type: Timestamp  
Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
CognitoMemberDefinition
Service: Amazon SageMaker Service

Identifies a Amazon Cognito user group. A user group can be used in on or more work teams.

Contents

ClientId
An identifier for an application client. You must create the app client ID using Amazon Cognito.

Type: String
Pattern: [\w+]+
Required: Yes

UserGroup
An identifier for a user group.

Type: String
Pattern: [\p{L}\p{M}\p{S}\p{N}\p{P}]+
Required: Yes

UserPool
An identifier for a user pool. The user pool must be in the same region as the service that you are calling.

Type: String
Pattern: [\w-]+[0-9a-zA-Z]+
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CollectionConfiguration
Service: Amazon SageMaker Service

Configuration information for tensor collections.

Contents

**CollectionName**

The name of the tensor collection. The name must be unique relative to other rule configuration names.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: .*

Required: No

**CollectionParameters**

Parameter values for the tensor collection. The allowed parameters are “name”, “include_regex”, “reduction_config”, “save_config”, “tensor_names”, and “save_histogram”.

Type: String to string map

Key Length Constraints: Minimum length of 1. Maximum length of 256.

Key Pattern: .*

Value Length Constraints: Maximum length of 256.

Value Pattern: .*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CompilationJobSummary
Service: Amazon SageMaker Service

A summary of a model compilation job.

Contents

CompilationEndTime
The time when the model compilation job completed.
Type: Timestamp
Required: No

CompilationJobArn
The Amazon Resource Name (ARN) of the model compilation job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]\{12\}:compilation-job/.*
Required: Yes

CompilationJobName
The name of the model compilation job that you want a summary for.
Type: String
Pattern: ^[a-zA-Z0-9\-\]*\-[a-zA-Z0-9\-]*$*
Required: Yes

CompilationJobStatus
The status of the model compilation job.
Type: String
Valid Values: INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED
Required: Yes

CompilationStartTime
The time when the model compilation job started.
Type: Timestamp
Required: No

CompilationTargetDevice
The type of device that the model will run on after compilation has completed.
Type: String
Valid Values: lambda | ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | ml_inf1 | jetson_tx1 | jetson_tx2 | jetson_nano | rasp3b | deplens | rk3399 | rk3288 | aisage | sbe_c | qcs605 | qcs603
**CreationTime**

The time when the model compilation job was created.

Type: Timestamp

Required: Yes

**LastModifiedTime**

The time when the model compilation job was last modified.

Type: Timestamp

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContainerDefinition
Service: Amazon SageMaker Service

Describes the container, as part of model definition.

Contents

ContainerHostname

This parameter is ignored for models that contain only a PrimaryContainer.

When a ContainerDefinition is part of an inference pipeline, the value of the parameter uniquely identifies the container for the purposes of logging and metrics. For information, see Use Logs and Metrics to Monitor an Inference Pipeline. If you don't specify a value for this parameter for a ContainerDefinition that is part of an inference pipeline, a unique name is automatically assigned based on the position of the ContainerDefinition in the pipeline. If you specify a value for the ContainerHostName for any ContainerDefinition that is part of an inference pipeline, you must specify a value for the ContainerHostName parameter of every ContainerDefinition in that pipeline.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\[a-zA-Z0-9\-\]]*$

Required: No

Environment

The environment variables to set in the Docker container. Each key and value in the Environment string to string map can have length of up to 1024. We support up to 16 entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: \[^a-zA-Z\-]([^a-zA-Z0-9\-\]/*

Value Length Constraints: Maximum length of 1024.

Value Pattern: \[^S\s]*

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored. If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both registry/repository[:tag] and registry/repository[@digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: \[^S\]+

Required: No
Mode

Whether the container hosts a single model or multiple models.

Type: String

Valid Values: SingleModel | MultiModel

Required: No

ModelDataUrl

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix). The S3 path is required for Amazon SageMaker built-in algorithms, but not if you use your own algorithms. For more information on built-in algorithms, see Common Parameters.

If you provide a value for this parameter, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provide. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS in an AWS Region in the AWS Identity and Access Management User Guide.

**Important**

If you use a built-in algorithm to create a model, Amazon SageMaker requires that you provide a S3 path to the model artifacts in ModelDataUrl.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)?([^/]+)$

Required: No

ModelPackageName

The name or Amazon Resource Name (ARN) of the model package to use to create the model.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:\{[a-z\-]*\}[/]?([a-zA-Z0-9\-]*)\{0,62\})(?!<:1-)$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContinuousParameterRange
Service: Amazon SageMaker Service

A list of continuous hyperparameters to tune.

Contents

MaxValue

The maximum value for the hyperparameter. The tuning job uses floating-point values between
MinValue value and this value for tuning.

Type: String
Length Constraints: Maximum length of 256.
Pattern: .*
Required: Yes

MinValue

The minimum value for the hyperparameter. The tuning job uses floating-point values between this
value and MaxValue for tuning.

Type: String
Length Constraints: Maximum length of 256.
Pattern: .*
Required: Yes

Name

The name of the continuous hyperparameter to tune.

Type: String
Length Constraints: Maximum length of 256.
Pattern: .*
Required: Yes

ScalingType

The scale that hyperparameter tuning uses to search the hyperparameter range. For information
about choosing a hyperparameter scale, see Hyperparameter Scaling. One of the following values:
Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.
Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale.
Logarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.
ReverseLogarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale.

Reverse logarithmic scaling works only for ranges that are entirely within the range 0<=x<1.0.

Type: String

Valid Values: Auto | Linear | Logarithmic | ReverseLogarithmic

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContinuousParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for a continuous hyperparameter.

Contents

MaxValue
The maximum floating-point value allowed.
Type: String
Length Constraints: Maximum length of 256.
Pattern: . *
Required: Yes

MinValue
The minimum floating-point value allowed.
Type: String
Length Constraints: Maximum length of 256.
Pattern: . *
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DataCaptureConfig
Service: Amazon SageMaker Service

Contents

CaptureContentTypeHeader
   Type: CaptureContentTypeHeader (p. 1306) object
   Required: No

CaptureOptions
   Type: Array of CaptureOption (p. 1307) objects
   Array Members: Minimum number of 1 item. Maximum number of 2 items.
   Required: Yes

DestinationS3Uri
   Type: String
   Length Constraints: Maximum length of 512.
   Pattern: ^(https|s3)://([^/])/(.*)$
   Required: Yes

EnableCapture
   Type: Boolean
   Required: No

InitialSamplingPercentage
   Type: Integer
   Valid Range: Minimum value of 0. Maximum value of 100.
   Required: Yes

KmsKeyId
   Type: String
   Length Constraints: Maximum length of 2048.
   Pattern: .*
   Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DataCaptureConfigSummary
Service: Amazon SageMaker Service

Contents

CaptureStatus
Type: String
Valid Values: Started | Stopped
Required: Yes

CurrentSamplingPercentage
Type: Integer
Valid Range: Minimum value of 0. Maximum value of 100.
Required: Yes

DestinationS3Uri
Type: String
Length Constraints: Maximum length of 512.
Pattern: ^(https|s3)://([^/])?(.*)$
Required: Yes

EnableCapture
Type: Boolean
Required: Yes

KmsKeyId
Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DataProcessing
Service: Amazon SageMaker Service

The data structure used to specify the data to be used for inference in a batch transform job and to associate the data that is relevant to the prediction results in the output. The input filter provided allows you to exclude input data that is not needed for inference in a batch transform job. The output filter provided allows you to include input data relevant to interpreting the predictions in the output from the job. For more information, see Associate Prediction Results with their Corresponding Input Records.

Contents

InputFilter

A JSONPath expression used to select a portion of the input data to pass to the algorithm. Use the InputFilter parameter to exclude fields, such as an ID column, from the input. If you want Amazon SageMaker to pass the entire input dataset to the algorithm, accept the default value $.

Examples: "$", "[1:]", ".features"

Type: String

Length Constraints: Minimum length of 0. Maximum length of 63.

Required: No

JoinSource

Specifies the source of the data to join with the transformed data. The valid values are None and Input. The default value is None, which specifies not to join the input with the transformed data. If you want the batch transform job to join the original input data with the transformed data, set JoinSource to Input.

For JSON or JSONLines objects, such as a JSON array, Amazon SageMaker adds the transformed data to the input JSON object in an attribute called SageMakerOutput. The joined result for JSON must be a key-value pair object. If the input is not a key-value pair object, Amazon SageMaker creates a new JSON file. In the new JSON file, and the input data is stored under the SageMakerInput key and the results are stored in SageMakerOutput.

For CSV files, Amazon SageMaker combines the transformed data with the input data at the end of the input data and stores it in the output file. The joined data has the joined input data followed by the transformed data and the output is a CSV file.

Type: String

Valid Values: Input | None

Required: No

OutputFilter

A JSONPath expression used to select a portion of the joined dataset to save in the output file for a batch transform job. If you want Amazon SageMaker to store the entire input dataset in the output file, leave the default value, $. If you specify indexes that aren't within the dimension size of the joined dataset, you get an error.

Examples: "$", "[0,5:]", "['id','SageMakerOutput']"

Type: String

Length Constraints: Minimum length of 0. Maximum length of 63.

Required: No
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DataSource
Service: Amazon SageMaker Service

Describes the location of the channel data.

Contents

FileSystemDataSource
The file system that is associated with a channel.

Type: FileSystemDataSource (p. 1352) object

Required: No

S3DataSource
The S3 location of the data source that is associated with a channel.

Type: S3DataSource (p. 1501) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DebugHookConfig
Service: Amazon SageMaker Service

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Contents

CollectionConfigurations
Configuration information for tensor collections.
Type: Array of CollectionConfiguration (p. 1318) objects
Array Members: Minimum number of 0 items. Maximum number of 20 items.
Required: No

HookParameters
Configuration information for the debug hook parameters.
Type: String to string map
Key Length Constraints: Minimum length of 1. Maximum length of 256.
Key Pattern: .*
Value Length Constraints: Maximum length of 256.
Value Pattern: .*
Required: No

LocalPath
Path to local storage location for tensors. Defaults to /opt/ml/output/tensors/.
Type: String
Length Constraints: Maximum length of 4096.
Pattern: .*
Required: No

S3OutputPath
Path to Amazon S3 storage location for tensors.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3)://([^/]+)/?(.*)$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
DebugRuleConfiguration

Service: Amazon SageMaker Service

Configuration information for debugging rules.

Contents

InstanceType

The instance type to deploy for a training job.

Type: String

Valid Values: ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge
| ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge
| ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge
| ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge
| ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge
| ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.m5.large
| ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge
| ml.m5.24xlarge | ml.r5.large | ml.r5.xlarge | ml.r5.2xlarge | ml.r5.4xlarge
| ml.r5.8xlarge | ml.r5.12xlarge | ml.r5.16xlarge | ml.r5.24xlarge

Required: No

LocalPath

Path to local storage location for output of rules. Defaults to /opt/ml/processing/output/rule/.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: .*

Required: No

RuleConfigurationName

The name of the rule configuration. It must be unique relative to other rule configuration names.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: .*

Required: Yes

RuleEvaluatorImage

The Amazon Elastic Container (ECR) Image for the managed rule evaluation.

Type: String

Length Constraints: Maximum length of 255.

Pattern: .*

Required: Yes
RuleParameters

Runtime configuration for rule container.
Type: String to string map
Key Length Constraints: Minimum length of 1. Maximum length of 256.
Key Pattern: . *
Value Length Constraints: Maximum length of 256.
Value Pattern: . *
Required: No

S3OutputPath

Path to Amazon S3 storage location for rules.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3)://([^/]+)/(.*)$
Required: No

VolumeSizeInGB

The size, in GB, of the ML storage volume attached to the processing instance.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DebugRuleEvaluationStatus
Service: Amazon SageMaker Service

Information about the status of the rule evaluation.

Contents

LastModifiedTime
  Timestamp when the rule evaluation status was last modified.
  Type: Timestamp
  Required: No

RuleConfigurationName
  The name of the rule configuration
  Type: String
  Length Constraints: Minimum length of 1. Maximum length of 256.
  Pattern: .*
  Required: No

RuleEvaluationJobArn
  The Amazon Resource Name (ARN) of the rule evaluation job.
  Type: String
  Length Constraints: Maximum length of 256.
  Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:processing-job/.*
  Required: No

RuleEvaluationStatus
  Status of the rule evaluation.
  Type: String
  Valid Values: InProgress | NoIssuesFound | IssuesFound | Error | Stopping | Stopped
  Required: No

StatusDetails
  Details from the rule evaluation.
  Type: String
  Length Constraints: Maximum length of 1024.
  Pattern: .*
  Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**DeployedImage**

*Service: Amazon SageMaker Service*

Gets the Amazon EC2 Container Registry path of the docker image of the model that is hosted in this `ProductionVariant` (p. 1484).

If you used the `registry/repository[:tag]` form to specify the image path of the primary container when you created the model hosted in this `ProductionVariant`, the path resolves to a path of the form `registry/repository[@digest]`. A digest is a hash value that identifies a specific version of an image. For information about Amazon ECR paths, see Pulling an Image in the Amazon ECR User Guide.

**Contents**

**ResolutionTime**

The date and time when the image path for the model resolved to the `ResolvedImage`

Type: Timestamp

Required: No

**ResolvedImage**

The specific digest path of the image hosted in this `ProductionVariant`.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[^\s]+`

Required: No

**SpecifiedImage**

The image path you specified when you created the model.

Type: String

Length Constraints: Maximum length of 255.

Pattern: `[^\s]+`

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**DesiredWeightAndCapacity**

Service: Amazon SageMaker Service

Specifies weight and capacity values for a production variant.

**Contents**

**DesiredInstanceCount**

The variant's capacity.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**DesiredWeight**

The variant's weight.

Type: Float

Valid Range: Minimum value of 0.

Required: No

**VariantName**

The name of the variant to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9][-*[a-zA-Z0-9]]*$`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DomainDetails
Service: Amazon SageMaker Service

The domain's details.

Contents

CreationTime
The creation time.
Type: Timestamp
Required: No

DomainArn
The domain's Amazon Resource Name (ARN).
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:domain/.*
Required: No

DomainId
The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: No

DomainName
The domain name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9\-]*(\-[a-zA-Z0-9]\\)*
Required: No

LastModifiedTime
The last modified time.
Type: Timestamp
Required: No

Status
The status.
Type: String
Valid Values: Deleting | Failed | InService | Pending
Required: No

**Url**

The domain's URL.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
EndpointConfigSummary
Service: Amazon SageMaker Service

Provides summary information for an endpoint configuration.

Contents

CreationTime

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

Required: Yes

EndpointConfigArn

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint-config/.*

Required: Yes

EndpointConfigName

The name of the endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-][\-]*[a-zA-Z0-9\-]*$*

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**EndpointInput**
Service: Amazon SageMaker Service

Input object for the endpoint

**Contents**

**EndpointName**

An endpoint in customer's account which has enabled DataCaptureConfig enabled.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\[0-9]](-*[a-zA-Z0-9\-\[0-9]])*

Required: Yes

**LocalPath**

Path to the filesystem where the endpoint data is available to the container.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

**S3DataDistributionType**

Whether input data distributed in Amazon S3 is fully replicated or sharded by an S3 key. Defaults to FullyReplicated

Type: String

Valid Values: FullyReplicated | ShardedByS3Key

Required: No

**S3InputMode**

Whether the Pipe or File is used as the input mode for transferring data for the monitoring job. Pipe mode is recommended for large datasets. File mode is useful for small files that fit in memory. Defaults to File.

Type: String

Valid Values: Pipe | File

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
• AWS SDK for Ruby V2
EndpointSummary
Service: Amazon SageMaker Service

Provides summary information for an endpoint.

Contents

CreationTime
A timestamp that shows when the endpoint was created.
Type: Timestamp
Required: Yes

EndpointArn
The Amazon Resource Name (ARN) of the endpoint.
Type: String
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:endpoint/.*
Required: Yes

EndpointName
The name of the endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9\-](\*[^a-zA-Z0-9\-])*
Required: Yes

EndpointStatus
The status of the endpoint.
• OutOfService: Endpoint is not available to take incoming requests.
• Creating: CreateEndpoint (p. 875) is executing.
• Updating: UpdateEndpoint (p. 1233) or UpdateEndpointWeightsAndCapacities (p. 1235) is executing.
• SystemUpdating: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
• RollingBack: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an InService status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an UpdateEndpointWeightsAndCapacities (p. 1235) call or when the UpdateEndpointWeightsAndCapacities (p. 1235) operation is called explicitly.
• InService: Endpoint is available to process incoming requests.
• Deleting: DeleteEndpoint (p. 963) is executing.
• Failed: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 1014) for information about the failure. DeleteEndpoint (p. 963) is the only operation that can be performed on a failed endpoint.
To get a list of endpoints with a specified status, use the `ListEndpoints:StatusEquals` (p. 1122) filter.

**Type:** String

**Valid Values:** OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed

**Required:** Yes

**LastModifiedTime**

A timestamp that shows when the endpoint was last modified.

**Type:** Timestamp

**Required:** Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Experiment
Service: Amazon SageMaker Service

A summary of the properties of an experiment as returned by the Search (p. 1196) API.

Contents

CreatedBy

Information about the user who created or modified an experiment, trial, or trial component.

Type: UserContext (p. 1572) object

Required: No

CreationTime

When the experiment was created.

Type: Timestamp

Required: No

Description

The description of the experiment.

Type: String

Length Constraints: Maximum length of 3072.

Pattern: .*

Required: No

DisplayName

The name of the experiment as displayed. If DisplayName isn't specified, ExperimentName is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* $a

Required: No

ExperimentArn

The Amazon Resource Name (ARN) of the experiment.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment/.*

Required: No

ExperimentName

The name of the experiment.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: No

**LastModifiedBy**

Information about the user who created or modified an experiment, trial, or trial component.
Type: UserContext (p. 1572) object
Required: No

**LastModifiedTime**

When the experiment was last modified.
Type: Timestamp
Required: No

**Source**

The source of the experiment.
Type: ExperimentSource (p. 1349) object
Required: No

**Tags**

The list of tags that are associated with the experiment. You can use Search (p. 1196) API to search on the tags.
Type: Array of Tag (p. 1517) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ExperimentConfig
Service: Amazon SageMaker Service

Configuration for the experiment.

Contents

ExperimentName
The name of the experiment.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: No

TrialComponentDisplayName
Display name for the trial component.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: No

TrialName
The name of the trial.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ExperimentSource**
Service: Amazon SageMaker Service

The source of the experiment.

**Contents**

**SourceArn**

The Amazon Resource Name (ARN) of the source.

Type: String

Length Constraints: Maximum length of 256.

Pattern: \*arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:.*

Required: Yes

**SourceType**

The source type.

Type: String

Length Constraints: Maximum length of 128.

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ExperimentSummary
Service: Amazon SageMaker Service

A summary of the properties of an experiment. To get the complete set of properties, call the DescribeExperiment (p. 1018) API and provide the ExperimentName.

Contents

CreationTime
When the experiment was created.
Type: Timestamp
Required: No

DisplayName
The name of the experiment as displayed. If DisplayName isn't specified, ExperimentName is displayed.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: No

ExperimentArn
The Amazon Resource Name (ARN) of the experiment.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment/.*  
Required: No

ExperimentName
The name of the experiment.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: No

ExperimentSource
The source of the experiment.
Type: ExperimentSource (p. 1349) object
Required: No

LastModifiedTime
When the experiment was last modified.
Type: Timestamp
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**FileSystemDataSource**

Service: Amazon SageMaker Service

Specifies a file system data source for a channel.

**Contents**

**DirectoryPath**

The full path to the directory to associate with the channel.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: .*

Required: Yes

**FileSystemAccessMode**

The access mode of the mount of the directory associated with the channel. A directory can be mounted either in **ro** (read-only) or **rw** (read-write) mode.

Type: String

Valid Values: **rw** | **ro**

Required: Yes

**FileSystemId**

The file system id.

Type: String

Length Constraints: Minimum length of 11.

Pattern: .*

Required: Yes

**FileSystemType**

The file system type.

Type: String

Valid Values: **EFS** | **FSxLustre**

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Filter
Service: Amazon SageMaker Service

A conditional statement for a search expression that includes a resource property, a Boolean operator, and a value.

If you don't specify an Operator and a Value, the filter searches for only the specified property. For example, defining a Filter for the FailureReason for the TrainingJob Resource searches for training job objects that have a value in the FailureReason field.

If you specify a Value, but not an Operator, Amazon SageMaker uses the equals operator as the default.

In search, there are several property types:

Metrics

To define a metric filter, enter a value using the form "Metrics.<name>", where <name> is a metric name. For example, the following filter searches for training jobs with an "accuracy" metric greater than "0.9":

```json
{
    "Name": "Metrics.accuracy",
    "Operator": "GREATER_THAN",
    "Value": "0.9"
}
```

HyperParameters

To define a hyperparameter filter, enter a value with the form "HyperParameters.<name>". Decimal hyperparameter values are treated as a decimal in a comparison if the specified Value is also a decimal value. If the specified Value is an integer, the decimal hyperparameter values are treated as integers. For example, the following filter is satisfied by training jobs with a "learning_rate" hyperparameter that is less than "0.5":

```json
{
    "Name": "HyperParameters.learning_rate",
    "Operator": "LESS_THAN",
    "Value": "0.5"
}
```

Tags

To define a tag filter, enter a value with the form "Tags.<key>".

Contents

Name

A property name. For example, TrainingJobName. For the list of valid property names returned in a search result for each supported resource, see TrainingJob (p. 1520) properties. You must specify a valid property name for the resource.
Operator

A Boolean binary operator that is used to evaluate the filter. The operator field contains one of the following values:

Equals

The specified resource in Name equals the specified Value.

NotEquals

The specified resource in Name does not equal the specified Value.

GreaterThan

The specified resource in Name is greater than the specified Value. Not supported for text-based properties.

GreaterThanOrEqualTo

The specified resource in Name is greater than or equal to the specified Value. Not supported for text-based properties.

LessThan

The specified resource in Name is less than the specified Value. Not supported for text-based properties.

LessThanOrEqualTo

The specified resource in Name is less than or equal to the specified Value. Not supported for text-based properties.

Contains

Only supported for text-based properties. The word-list of the property contains the specified Value. A SearchExpression can include only one Contains operator.

If you have specified a filter Value, the default is Equals.

Value

A value used with Resource and Operator to determine if objects satisfy the filter’s condition. For numerical properties, Value must be an integer or floating-point decimal. For timestamp properties, Value must be an ISO 8601 date-time string of the following format: YYYY-mm-dd'T'HH:MM:SS.
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
FinalAutoMLJobObjectiveMetric
Service: Amazon SageMaker Service

The candidate result from a job.

Contents

MetricName
The name of the metric.
Type: String
Valid Values: Accuracy | MSE | F1 | F1macro
Required: Yes

Type
The metric type used.
Type: String
Valid Values: Maximize | Minimize
Required: No

Value
The value of the metric.
Type: Float
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
FinalHyperParameterTuningJobObjectiveMetric

Service: Amazon SageMaker Service

Shows the final value for the objective metric for a training job that was launched by a hyperparameter
tuning job. You define the objective metric in the HyperParameterTuningJobObjective parameter
of HyperParameterTuningJobConfig (p. 1389).

Contents

MetricName

The name of the objective metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: Yes

Type

Whether to minimize or maximize the objective metric. Valid values are Minimize and Maximize.

Type: String

Valid Values: Maximize | Minimize

Required: No

Value

The value of the objective metric.

Type: Float

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
FlowDefinitionOutputConfig
Service: Amazon SageMaker Service
Contains information about where human output will be stored.

Contents

KmsKeyId
The Amazon Key Management Service (KMS) key ID for server-side encryption.
Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: No

S3OutputPath
The Amazon S3 path where the object containing human output will be made available.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3)//([^/]+)/?(.*)$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
FlowDefinitionSummary
Service: Amazon SageMaker Service

Contains summary information about the flow definition.

Contents

CreationTime
The timestamp when SageMaker created the flow definition.
Type: Timestamp
Required: Yes

FailureReason
The reason why the flow definition creation failed. A failure reason is returned only when the flow definition status is Failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

FlowDefinitionArn
The Amazon Resource Name (ARN) of the flow definition.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:flow-definition/.*
Required: Yes

FlowDefinitionName
The name of the flow definition.
Type: String
Pattern: ^[a-z0-9\-]{1,63}$
Required: Yes

FlowDefinitionStatus
The status of the flow definition. Valid values:
Type: String
Valid Values: Initializing | Active | Failed | Deleting | Deleted
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
GitConfig
Service: Amazon SageMaker Service

Specifies configuration details for a Git repository in your AWS account.

Contents

Branch

The default branch for the Git repository.

Type: String


Pattern: \^[^ ~^:?*\[\] ]+

Required: No

RepositoryUrl

The URL where the Git repository is located.

Type: String

Pattern: ^https://([^/]+)/?(.*)$

Required: Yes

SecretArn

The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of AWSCURRENT and must be in the following format:

{"username": UserName, "password": Password}

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:secretsmanager:[a-z0-9\-]*:[0-9]{12}:secret:.*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
GitConfigForUpdate
Service: Amazon SageMaker Service

Specifies configuration details for a Git repository when the repository is updated.

Contents

SecretArn

The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of AWSCURRENT and must be in the following format:

{"username": Username, "password": Password}

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:secretsmanager:[a-z0-9\-]*:[0-9]{12}:secret:.*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanLoopActivationConditionsConfig
Service: Amazon SageMaker Service

Defines under what conditions SageMaker creates a human loop. Used within CreateFlowDefinition (p. 885).

Contents

HumanLoopActivationConditions

JSON expressing use-case specific conditions declaratively. If any condition is matched, atomic tasks are created against the configured work team. The set of conditions is different for Rekognition and Textract.

Type: String

Length Constraints: Maximum length of 10240.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanLoopActivationConfig

Service: Amazon SageMaker Service

Provides information about how and under what conditions SageMaker creates a human loop. If HumanLoopActivationConfig is not given, then all requests go to humans.

Contents

HumanLoopActivationConditionsConfig

Container structure for defining under what conditions SageMaker creates a human loop.

Type: HumanLoopActivationConditionsConfig (p. 1364) object

Required: Yes

HumanLoopRequestSource

Container for configuring the source of human task requests.

Type: HumanLoopRequestSource (p. 1371) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanLoopConfig
Service: Amazon SageMaker Service

Describes the work to be performed by human workers.

Contents

HumanTaskUiArn

The Amazon Resource Name (ARN) of the human task user interface.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:human-task-ui/.*

Required: Yes

PublicWorkforceTaskPrice

Defines the amount of money paid to an Amazon Mechanical Turk worker for each task performed.

Use one of the following prices for bounding box tasks. Prices are in US dollars and should be based on the complexity of the task; the longer it takes in your initial testing, the more you should offer.

- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for image classification, text classification, and custom tasks. Prices are in US dollars.

- 0.012
- 0.024
- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
Use one of the following prices for semantic segmentation tasks. Prices are in US dollars.
• 0.840
• 0.960
• 1.080
• 1.200

Use one of the following prices for Textract AnalyzeDocument Important Form Key Amazon Augmented AI review tasks. Prices are in US dollars.
• 2.400
• 2.280
• 2.160
• 2.040
• 1.920
• 1.800
• 1.680
• 1.560
• 1.440
• 1.320
• 1.200
• 1.080
• 0.960
• 0.840
• 0.720
• 0.600
• 0.480
• 0.360
• 0.240
• 0.120
• 0.072
• 0.060
• 0.048
• 0.036
• 0.024
• 0.012

Use one of the following prices for Rekognition DetectModerationLabels Amazon Augmented AI review tasks. Prices are in US dollars.
• 1.200
• 1.080
• 0.960
• 0.840
• 0.720
Use one of the following prices for Amazon Augmented AI custom human review tasks. Prices are in US dollars.

- 1.200
- 1.080
- 0.960
- 0.840
- 0.720
- 0.600
- 0.480
- 0.360
- 0.240
- 0.120
- 0.072
- 0.060
- 0.048
- 0.036
- 0.024
- 0.012

Type: `PublicWorkforceTaskPrice` (p. 1490) object

Required: No

**TaskAvailabilityLifetimeInSeconds**

The length of time that a task remains available for labeling by human workers.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 864000.

Required: No

**TaskCount**

The number of human tasks.

Type: Integer


Required: Yes
**TaskDescription**

A description for the human worker task.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: Yes

**TaskKeywords**

Keywords used to describe the task so that workers can discover the task.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 5 items.


Pattern: ^[A-Za-z0-9]+( [A-Za-z0-9]+)*$

Required: No

**TaskTimeLimitInSeconds**

The amount of time that a worker has to complete a task.

Type: Integer


Required: No

**TaskTitle**

A title for the human worker task.

Type: String


Pattern: ^[^\t\n\r \u0009-\u001f ]*$

Required: Yes

**WorkteamArn**

Amazon Resource Name (ARN) of a team of workers.

Type: String

Length Constraints: Maximum length of 256.

Pattern:arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanLoopRequestSource
Service: Amazon SageMaker Service

Container for configuring the source of human task requests.

Contents

AwsManagedHumanLoopRequestSource

Specifies whether Amazon Rekognition or Amazon Textract are used as the integration source. The default field settings and JSON parsing rules are different based on the integration source. Valid values:

Type: String

Valid Values: AWS/Rekognition/DetectModerationLabels/Image/V3 | AWS/Textextract/AnalyzeDocument/Forms/V1

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanTaskConfig
Service: Amazon SageMaker Service

Information required for human workers to complete a labeling task.

Contents

AnnotationConsolidationConfig

Configures how labels are consolidated across human workers.

Type: AnnotationConsolidationConfig (p. 1282) object

Required: Yes

MaxConcurrentTaskCount

Defines the maximum number of data objects that can be labeled by human workers at the same time. Also referred to as batch size. Each object may have more than one worker at one time. The default value is 1000 objects.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

NumberOfHumanWorkersPerDataObject

The number of human workers that will label an object.

Type: Integer


Required: Yes

PreHumanTaskLambdaArn

The Amazon Resource Name (ARN) of a Lambda function that is run before a data object is sent to a human worker. Use this function to provide input to a custom labeling job.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

**US East (Northern Virginia) (us-east-1):**
- arn:aws:lambda:us-east-1:432418664414:function:PRE-BoundingBox
- arn:aws:lambda:us-east-1:432418664414:function:PRE-TextMultiClass
- arn:aws:lambda:us-east-1:432418664414:function:PRE-VerificationBoundingBox
- arn:aws:lambda:us-east-1:432418664414:function:PRE-AdjustmentBoundingBox

**US East (Ohio) (us-east-2):**
• arn:aws:lambda:us-east-2:266458841044:function:PRE-TextMultiClass
• arn:aws:lambda:us-east-2:266458841044:function:PRE-VerificationBoundingBox
• arn:aws:lambda:us-east-2:266458841044:function:PRE-AdjustmentBoundingBox

US West (Oregon) (us-west-2):
• arn:aws:lambda:us-west-2:081040173940:function:PRE-BoundingBox
• arn:aws:lambda:us-west-2:081040173940:function:PRE-TextMultiClass
• arn:aws:lambda:us-west-2:081040173940:function:PRE-VerificationBoundingBox
• arn:aws:lambda:us-west-2:081040173940:function:PRE-AdjustmentBoundingBox

Canada (Central) (ca-central-1):
• arn:aws:lambda:ca-central-1:918755190332:function:PRE-BoundingBox
• arn:aws:lambda:ca-central-1:918755190332:function:PRE-ImageMultiClass
• arn:aws:lambda:ca-central-1:918755190332:function:PRE-TextMultiClass
• arn:aws:lambda:ca-central-1:918755190332:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ca-central-1:918755190332:function:PRE-AdjustmentBoundingBox

EU (Ireland) (eu-west-1):
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-BoundingBox
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-ImageMultiClass
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-SemanticSegmentation
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-TextMultiClass
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-NamedEntityRecognition
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-VerificationBoundingBox
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-VerificationSemanticSegmentation
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-AdjustmentBoundingBox
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-AdjustmentSemanticSegmentation

EU (London) (eu-west-2):
• arn:aws:lambda:eu-west-2:487402164563:function:PRE-BoundingBox
• arn:aws:lambda:eu-west-2:487402164563:function:PRE-ImageMultiClass
• arn:aws:lambda:eu-west-2:487402164563:function:PRE-TextMultiClass
• arn:aws:lambda:eu-west-2:487402164563:function:PRE-VerificationBoundingBox
• arn:aws:lambda:eu-west-2:487402164563:function:PRE-AdjustmentBoundingBox

EU Frankfurt (eu-central-1):
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-BoundingBox
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-ImageMultiClass
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-SemanticSegmentation
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-TextMultiClass
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-VerificationBoundingBox
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-VerificationSemanticSegmentation
• arn:aws:lambda:eu-central-1:203001061592:function:PRE-AdjustmentBoundingBox

Asia Pacific (Tokyo) (ap-northeast-1):
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-BoundingBox
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-ImageMultiClass
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-SemanticSegmentation
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-TextMultiClass
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-VerificationSemanticSegmentation
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-AdjustmentBoundingBox
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-AdjustmentSemanticSegmentation

Asia Pacific (Seoul) (ap-northeast-2):
• arn:aws:lambda:ap-northeast-2:845288260483:function:PRE-BoundingBox
• arn:aws:lambda:ap-northeast-2:845288260483:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ap-northeast-2:845288260483:function:PRE-AdjustmentBoundingBox

Asia Pacific (Mumbai) (ap-south-1):
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-BoundingBox
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-ImageMultiClass
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-SemanticSegmentation
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-TextMultiClass
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-VerificationSemanticSegmentation
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-AdjustmentBoundingBox
• arn:aws:lambda:ap-south-1:565803892007:function:PRE-AdjustmentSemanticSegmentation

Asia Pacific (Singapore) (ap-southeast-1):
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-BoundingBox
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-ImageMultiClass
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-SemanticSegmentation
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-TextMultiClass
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-VerificationSemanticSegmentation
• arn:aws:lambda:ap-southeast-1:377565633583:function:PRE-AdjustmentBoundingBox

Asia Pacific (Sydney) (ap-southeast-2):
• arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-BoundingBox
• arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-VerificationBoundingBox
• arn:aws:lambda:ap-southeast-2:454466003867:function:PRE-AdjustmentBoundingBox

Type: String
Length Constraints: Maximum length of 2048.
Pattern: \w*:\w+:\d+:\d+:\w+:\w+:\d+:\w+(:\w+)?
Required: Yes

**PublicWorkforceTaskPrice**

The price that you pay for each task performed by an Amazon Mechanical Turk worker.

Type: [PublicWorkforceTaskPrice](p. 1490) object
Required: No

**TaskAvailabilityLifetimeInSeconds**

The length of time that a task remains available for labeling by human workers. *If you choose the Amazon Mechanical Turk workforce, the maximum is 12 hours (43200).* The default value is 864000 seconds (1 day). For private and vendor workforces, the maximum is as listed.

Type: Integer
Valid Range: Minimum value of 60. Maximum value of 864000.
Required: No

**TaskDescription**

A description of the task for your human workers.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: .+
Required: Yes

**TaskKeywords**

Keywords used to describe the task so that workers on Amazon Mechanical Turk can discover the task.

Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 5 items.


Pattern: ^[A-Za-z0-9]+( [A-Za-z0-9]+)*$

Required: No

TaskTimeLimitInSeconds

The amount of time that a worker has to complete a task.

Type: Integer


Required: Yes

TaskTitle

A title for the task for your human workers.

Type: String


Pattern: ^[	
\n\r -\uD7FF\uE000-\uFFFD]*$

Required: Yes

UiConfig

Information about the user interface that workers use to complete the labeling task.

Type: UiConfig (p. 1568) object

Required: Yes

WorkteamArn

The Amazon Resource Name (ARN) of the work team assigned to complete the tasks.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanTaskUiSummary
Service: Amazon SageMaker Service

Container for human task user interface information.

Contents

CreationTime
A timestamp when SageMaker created the human task user interface.

Type: Timestamp
Required: Yes

HumanTaskUiArn
The Amazon Resource Name (ARN) of the human task user interface.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:human-task-ui/.*
Required: Yes

HumanTaskUiName
The name of the human task user interface.

Type: String
Pattern: ^[a-z0-9](\*[a-z0-9])*
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterAlgorithmSpecification

Service: Amazon SageMaker Service

Specifies which training algorithm to use for training jobs that a hyperparameter tuning job launches and the metrics to monitor.

Contents

AlgorithmName

The name of the resource algorithm to use for the hyperparameter tuning job. If you specify a value for this parameter, do not specify a value for TrainingImage.

Type: String


Pattern: (arn:aws[a-zA-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-zA-z\-]*\//)?([a-zA-Z0-9]\[a-zA-Z0-9-]{0,62})?([a-zA-Z0-9]\[a-zA-Z0-9-\]\{0,62\})\(?<!\-\)#

Required: No

MetricDefinitions

An array of MetricDefinition (p. 1425) objects that specify the metrics that the algorithm emits.

Type: Array of MetricDefinition (p. 1425) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about Docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters. Amazon SageMaker supports both registry/repository[:tag] and registry/repository@[digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: .*

Required: No

TrainingInputMode

The input mode that the algorithm supports: File or Pipe. In File input mode, Amazon SageMaker downloads the training data from Amazon S3 to the storage volume that is attached to the training instance and mounts the directory to the Docker volume for the training container. In Pipe input mode, Amazon SageMaker streams data directly from Amazon S3 to the container.

If you specify File mode, make sure that you provision the storage volume that is attached to the training instance with enough capacity to accommodate the training data downloaded from Amazon S3, the model artifacts, and intermediate information.

For more information about input modes, see Algorithms.

Type: String
Valid Values: Pipe | File
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterSpecification
Service: Amazon SageMaker Service

Defines a hyperparameter to be used by an algorithm.

Contents

DefaultValue

The default value for this hyperparameter. If a default value is specified, a hyperparameter cannot be required.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

Description

A brief description of the hyperparameter.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

IsRequired

Indicates whether this hyperparameter is required.

Type: Boolean

Required: No

IsTunable

Indicates whether this hyperparameter is tunable in a hyperparameter tuning job.

Type: Boolean

Required: No

Name

The name of this hyperparameter. The name must be unique.

Type: String

Length Constraints: Maximum length of 256.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: Yes

Range

The allowed range for this hyperparameter.

Type: ParameterRange (p. 1468) object
Required: No

Type

The type of this hyperparameter. The valid types are Integer, Continuous, Categorical, and FreeText.

Type: String

Valid Values: Integer | Continuous | Categorical | FreeText

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTrainingJobDefinition
Service: Amazon SageMaker Service

Defines the training jobs launched by a hyperparameter tuning job.

Contents

AlgorithmSpecification

The HyperParameterAlgorithmSpecification (p. 1379) object that specifies the resource algorithm to use for the training jobs that the tuning job launches.

Type: HyperParameterAlgorithmSpecification (p. 1379) object

Required: Yes

CheckpointConfig

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

Required: No

DefinitionName

The job definition name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

EnableInterContainerTrafficEncryption

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableManagedSpotTraining

A Boolean indicating whether managed spot training is enabled (True) or not (False).

Type: Boolean

Required: No

EnableNetworkIsolation

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If network isolation is used for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Type: Boolean
HyperParameterRanges

Specifies ranges of integer, continuous, and categorical hyperparameters that a hyperparameter tuning job searches. The hyperparameter tuning job launches training jobs with hyperparameter values within these ranges to find the combination of values that result in the training job with the best performance as measured by the objective metric of the hyperparameter tuning job.

Note
You can specify a maximum of 20 hyperparameters that a hyperparameter tuning job can search over. Every possible value of a categorical parameter range counts against this limit.

Type: ParameterRanges (p. 1469) object
Required: No

InputDataConfig

An array of Channel (p. 1310) objects that specify the input for the training jobs that the tuning job launches.

Type: Array of Channel (p. 1310) objects
Array Members: Minimum number of 1 item. Maximum number of 20 items.
Required: No

OutputDataConfig

Specifies the path to the Amazon S3 bucket where you store model artifacts from the training jobs that the tuning job launches.

Type: OutputDataConfig (p. 1466) object
Required: Yes

ResourceConfig

The resources, including the compute instances and storage volumes, to use for the training jobs that the tuning job launches.

Storage volumes store model artifacts and incremental states. Training algorithms might also use storage volumes for scratch space. If you want Amazon SageMaker to use the storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 1496) object
Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role associated with the training jobs that the tuning job launches.

Type: String
Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9_+\-,@\-_\/%]*$-
Required: Yes

StaticHyperParameters

Specifies the values of hyperparameters that do not change for the tuning job.
Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: . *

Value Length Constraints: Maximum length of 256.

Value Pattern: . *

Required: No

StoppingCondition

Specifies a limit to how long a model hyperparameter training job can run. It also specifies how long you are willing to wait for a managed spot training job to complete. When the job reaches the a limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

Type: StoppingCondition (p. 1513) object

Required: Yes

TuningObjective

Defines the objective metric for a hyperparameter tuning job. Hyperparameter tuning uses the value of this metric to evaluate the training jobs it launches, and returns the training job that results in either the highest or lowest value for this metric, depending on the value you specify for the Type parameter.

Type: HyperParameterTuningJobObjective (p. 1391) object

Required: No

VpcConfig

The VpcConfig (p. 1577) object that specifies the VPC that you want the training jobs that this hyperparameter tuning job launches to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTrainingJobSummary
Service: Amazon SageMaker Service

Specifies summary information about a training job.

Contents

CreationTime
The date and time that the training job was created.
Type: Timestamp
Required: Yes

FailureReason
The reason that the training job failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

FinalHyperParameterTuningJobObjectiveMetric
The FinalHyperParameterTuningJobObjectiveMetric (p. 1358) object that specifies the value of the objective metric of the tuning job that launched this training job.
Type: FinalHyperParameterTuningJobObjectiveMetric (p. 1358) object
Required: No

ObjectiveStatus
The status of the objective metric for the training job:
• Succeeded: The final objective metric for the training job was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.
• Pending: The training job is in progress and evaluation of its final objective metric is pending.
• Failed: The final objective metric for the training job was not evaluated, and was not used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.
Type: String
Valid Values: Succeeded | Pending | Failed
Required: No

TrainingEndTime
Specifies the time when the training job ends on training instances. You are billed for the time interval between the value of TrainingStartTime and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.
Type: Timestamp
Required: No

TrainingJobArn
The Amazon Resource Name (ARN) of the training job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:training-job/.*
Required: Yes

**TrainingJobDefinitionName**
The training job definition name.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$  
Required: No

**TrainingJobName**
The name of the training job.
Type: String
Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$  
Required: Yes

**TrainingJobStatus**
The status of the training job.
Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: Yes

**TrainingStartTime**
The date and time that the training job started.
Type: Timestamp
Required: No

**TunedHyperParameters**
A list of the hyperparameters for which you specified ranges to search.
Type: String to string map
Key Length Constraints: Maximum length of 256.
Key Pattern: .*
Value Length Constraints: Maximum length of 256.
Value Pattern: .*
Required: Yes
**TuningJobName**

The HyperParameter tuning job that launched the training job.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*?

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobConfig

Service: Amazon SageMaker Service

Configures a hyperparameter tuning job.

Contents

HyperParameterTuningJobObjective

The HyperParameterTuningJobObjective (p. 1391) object that specifies the objective metric for this tuning job.

Type: HyperParameterTuningJobObjective (p. 1391) object

Required: No

ParameterRanges

The ParameterRanges (p. 1469) object that specifies the ranges of hyperparameters that this tuning job searches.

Type: ParameterRanges (p. 1469) object

Required: No

ResourceLimits

The ResourceLimits (p. 1498) object that specifies the maximum number of training jobs and parallel training jobs for this tuning job.

Type: ResourceLimits (p. 1498) object

Required: Yes

Strategy

Specifies how hyperparameter tuning chooses the combinations of hyperparameter values to use for the training job it launches. To use the Bayesian search strategy, set this to Bayesian. To randomly search, set it to Random. For information about search strategies, see How Hyperparameter Tuning Works.

Type: String

Valid Values: Bayesian | Random

Required: Yes

TrainingJobEarlyStoppingType

Specifies whether to use early stopping for training jobs launched by the hyperparameter tuning job. This can be one of the following values (the default value is OFF):

OFF

Training jobs launched by the hyperparameter tuning job do not use early stopping.

AUTO

Amazon SageMaker stops training jobs launched by the hyperparameter tuning job when they are unlikely to perform better than previously completed training jobs. For more information, see Stop Training Jobs Early.

Type: String

Valid Values: Off | Auto
Required: No

**TuningJobCompletionCriteria**

The tuning job's completion criteria.

Type: TuningJobCompletionCriteria (p. 1567) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobObjective
Service: Amazon SageMaker Service

Defines the objective metric for a hyperparameter tuning job. Hyperparameter tuning uses the value of this metric to evaluate the training jobs it launches, and returns the training job that results in either the highest or lowest value for this metric, depending on the value you specify for the Type parameter.

Contents

MetricName

The name of the metric to use for the objective metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: Yes

Type

Whether to minimize or maximize the objective metric.

Type: String

Valid Values: Maximize | Minimize

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobSummary
Service: Amazon SageMaker Service

Provides summary information about a hyperparameter tuning job.

Contents

CreationTime
The date and time that the tuning job was created.
Type: Timestamp
Required: Yes

HyperParameterTuningEndTime
The date and time that the tuning job ended.
Type: Timestamp
Required: No

HyperParameterTuningJobArn
The Amazon Resource Name (ARN) of the tuning job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: \[arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*\]
Required: Yes

HyperParameterTuningJobName
The name of the tuning job.
Type: String
Pattern: ^[a-zA-Z0-9-]*\*[a-zA-Z0-9-]*\*
Required: Yes

HyperParameterTuningJobStatus
The status of the tuning job.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: Yes

LastModifiedTime
The date and time that the tuning job was modified.
Type: Timestamp
Required: No
**ObjectiveStatusCounters**

The ObjectiveStatusCounters (p. 1464) object that specifies the numbers of training jobs, categorized by objective metric status, that this tuning job launched.

Type: ObjectiveStatusCounters (p. 1464) object

Required: Yes

**ResourceLimits**

The ResourceLimits (p. 1498) object that specifies the maximum number of training jobs and parallel training jobs allowed for this tuning job.

Type: ResourceLimits (p. 1498) object

Required: No

**Strategy**

Specifies the search strategy hyperparameter tuning uses to choose which hyperparameters to use for each iteration. Currently, the only valid value is Bayesian.

Type: String

Valid Values: Bayesian | Random

Required: Yes

**TrainingJobStatusCounters**

The TrainingJobStatusCounters (p. 1529) object that specifies the numbers of training jobs, categorized by status, that this tuning job launched.

Type: TrainingJobStatusCounters (p. 1529) object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobWarmStartConfig
Service: Amazon SageMaker Service

Specifies the configuration for a hyperparameter tuning job that uses one or more previous hyperparameter tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric, and the training job that performs the best is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

Note
All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.

Contents

ParentHyperParameterTuningJobs

An array of hyperparameter tuning jobs that are used as the starting point for the new hyperparameter tuning job. For more information about warm starting a hyperparameter tuning job, see Using a Previous Hyperparameter Tuning Job as a Starting Point.

Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Type: Array of ParentHyperParameterTuningJob (p. 1471) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

WarmStartType

Specifies one of the following:

IDENTICAL_DATA_AND_ALGORITHM

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

TRANSFER_LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. The training image can also be a different version from the version used in the parent hyperparameter tuning job. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

Type: String

Valid Values: IdenticalDataAndAlgorithm | TransferLearning
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InferenceSpecification
Service: Amazon SageMaker Service

Defines how to perform inference generation after a training job is run.

Contents

Containers

The Amazon ECR registry path of the Docker image that contains the inference code.

Type: Array of ModelPackageContainerDefinition (p. 1427) objects

Array Members: Fixed number of 1 item.

Required: Yes

SupportedContentTypes

The supported MIME types for the input data.

Type: Array of strings

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

SupportedRealtimeInferenceInstanceTypes

A list of the instance types that are used to generate inferences in real-time.

Type: Array of strings

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge |
| ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge |
| ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge |
| ml.m5.12xlarge | ml.m5.24xlarge | ml.m5d.large | ml.m5d.xlarge |
| ml.m5d.2xlarge | ml.m5d.4xlarge | ml.m5d.12xlarge | ml.m5d.24xlarge |
| ml.c4.large | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge |
| ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge |
| ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge |
| ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.large |
| ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge |
| ml.c5d.18xlarge | ml.g4dn.xlarge | ml.g4dn.2xlarge | ml.g4dn.4xlarge |
| ml.g4dn.8xlarge | ml.g4dn.12xlarge | ml.g4dn.16xlarge | ml.r5.large |
| ml.r5.xlarge | ml.r5.2xlarge | ml.r5.4xlarge | ml.r5.12xlarge |
| ml.r5.24xlarge | ml.r5d.large | ml.r5d.xlarge | ml.r5d.2xlarge |
| ml.r5d.4xlarge | ml.r5d.12xlarge | ml.r5d.24xlarge | ml.inf1.xlarge |
| ml.inf1.2xlarge | ml.inf1.6xlarge | ml.inf1.24xlarge |

Required: Yes

SupportedResponseMIMETypes

The supported MIME types for the output data.

Type: Array of strings

Length Constraints: Maximum length of 1024.
Pattern: ^[-\w]+\/\./.+$  
Required: Yes  

**SupportedTransformInstanceTypes**  
A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

**Type:** Array of strings  

**Array Members:** Minimum number of 1 item.

**Valid Values:**  
ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge  
| ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge  
| ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge  
| ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge  

Required: Yes  

**See Also**  
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InputConfig
Service: Amazon SageMaker Service

Contains information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Contents
DataInputConfig

Specifies the name and shape of the expected data inputs for your trained model with a JSON dictionary form. The data inputs are InputConfig:Framework (p. 1399) specific.

- TensorFlow: You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, {"input": [1,1024,1024,3]}
    - If using the CLI, {"input": [1,1024,1024,3]}
  - Examples for two inputs:
    - If using the console, {"data1": [1,28,28,1], "data2": [1,28,28,1]}
    - If using the CLI, {"data1": [1,28,28,1], "data2": [1,28,28,1]}

- KERAS: You must specify the name and shape (NCHW format) of expected data inputs using a dictionary format for your trained model. Note that while Keras model artifacts should be uploaded in NHWC (channel-last) format, DataInputConfig should be specified in NCHW (channel-first) format. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, {"input_1": [1,3,224,224]}
    - If using the CLI, {"input_1": [1,3,224,224]}
  - Examples for two inputs:
    - If using the console, {"input_1": [1,3,224,224], "input_2": [1,3,224,224]}
    - If using the CLI, {"input_1": [1,3,224,224], "input_2": [1,3,224,224]}

- MXNET/ONNX: You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, {"data": [1,3,1024,1024]}
    - If using the CLI, {"data": [1,3,1024,1024]}
  - Examples for two inputs:
    - If using the console, {"var1": [1,1,28,28], "var2": [1,1,28,28]}
    - If using the CLI, {"var1": [1,1,28,28], "var2": [1,1,28,28]}

- PyTorch: You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
  - Examples for one input in dictionary format:
    - If using the console, {"input0": [1,3,224,224]}
    - If using the CLI, {"input0": [1,3,224,224]}
  - Example for one input in list format: [[1,3,224,224]]
  - Examples for two inputs in dictionary format:
    - If using the console, {"input0": [1,3,224,224], "input1": [1,3,224,224]}
- If using the CLI, `{"input0":[1,3,224,224], "input1":[1,3,224,224]}`
- Example for two inputs in list format: `[[1,3,224,224], [1,3,224,224]]`
- **XGBOOST**: input data name and shape are not needed.

**Type**: String

**Length Constraints**: Minimum length of 1. Maximum length of 1024.

**Pattern**: `\S\s+`

**Required**: Yes

### Framework

Identifies the framework in which the model was trained. For example: TENSORFLOW.

**Type**: String

**Valid Values**: TENSORFLOW | KERAS | MXNET | ONNX | PYTORCH | XGBOOST

**Required**: Yes

### S3Uri

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

**Type**: String

**Length Constraints**: Maximum length of 1024.

**Pattern**: `^(https|s3)://([^/]+)/?(.*)$`

**Required**: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**IntegerParameterRange**

Service: Amazon SageMaker Service

For a hyperparameter of the integer type, specifies the range that a hyperparameter tuning job searches.

**Contents**

**MaxValue**

The maximum value of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

**MinValue**

The minimum value of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

**Name**

The name of the hyperparameter to search.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

**ScalingType**

The scale that hyperparameter tuning uses to search the hyperparameter range. For information about choosing a hyperparameter scale, see [Hyperparameter Scaling](#). One of the following values:

- **Auto**
  
  Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

- **Linear**
  
  Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale.

- **Logarithmic**
  
  Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

  Logarithmic scaling works only for ranges that have only values greater than 0.

Type: String
Valid Values: Auto | Linear | Logarithmic | ReverseLogarithmic

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
IntegerParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for an integer hyperparameter.

Contents

MaxValue

The maximum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

MinValue

The minimum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
JupyterServerAppSettings
Service: Amazon SageMaker Service

Jupyter server's app settings.

Contents

DefaultResourceSpec

The instance type and quantity.

Type: ResourceSpec (p. 1499) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
KernelGatewayAppSettings
Service: Amazon SageMaker Service

The kernel gateway app settings.

Contents

DefaultResourceSpec

The instance type and quantity.

Type: ResourceSpec (p. 1499) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelCounters
Service: Amazon SageMaker Service

Provides a breakdown of the number of objects labeled.

Contents

FailedNonRetryableError
The total number of objects that could not be labeled due to an error.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

HumanLabeled
The total number of objects labeled by a human worker.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

MachineLabeled
The total number of objects labeled by automated data labeling.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

TotalLabeled
The total number of objects labeled.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

Unlabeled
The total number of objects not yet labeled.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**LabelCountersForWorkteam**  
Service: Amazon SageMaker Service

Provides counts for human-labeled tasks in the labeling job.

**Contents**

**HumanLabeled**

The total number of data objects labeled by a human worker.

Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**PendingHuman**

The total number of data objects that need to be labeled by a human worker.

Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**Total**

The total number of tasks in the labeling job.

Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**LabelingJobAlgorithmsConfig**  
Service: Amazon SageMaker Service

Provides configuration information for auto-labeling of your data objects. A LabelingJobAlgorithmsConfig object must be supplied in order to use auto-labeling.

**Contents**

**InitialActiveLearningModelArn**

At the end of an auto-label job Amazon SageMaker Ground Truth sends the Amazon Resource Name (ARN) of the final model used for auto-labeling. You can use this model as the starting point for subsequent similar jobs by providing the ARN of the model here.

Type: String


Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*`

Required: No

**LabelingJobAlgorithmSpecificationArn**

Specifies the Amazon Resource Name (ARN) of the algorithm used for auto-labeling. You must select one of the following ARNs:

- **Image classification**
  

- **Text classification**
  

- **Object detection**
  

- **Semantic Segmentation**
  

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:*`

Required: Yes

**LabelingJobResourceConfig**

Provides configuration information for a labeling job.

Type: LabelingJobResourceConfig (p. 1417) object

Required: No
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobDataAttributes
Service: Amazon SageMaker Service

Attributes of the data specified by the customer. Use these to describe the data to be labeled.

Contents

ContentClassifiers

Declares that your content is free of personally identifiable information or adult content. Amazon SageMaker may restrict the Amazon Mechanical Turk workers that can view your task based on this information.

Type: Array of strings

Array Members: Maximum number of 256 items.

Valid Values: FreeOfPersonallyIdentifiableInformation | FreeOfAdultContent

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobDataSource
Service: Amazon SageMaker Service

Provides information about the location of input data.

Contents

S3DataSource

The Amazon S3 location of the input data objects.

Type: LabelingJobS3DataSource (p. 1418) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**LabelingJobForWorkteamSummary**

Service: Amazon SageMaker Service

Provides summary information for a work team.

**Contents**

**CreationTime**

The date and time that the labeling job was created.

Type: Timestamp

Required: Yes

**JobReferenceCode**

A unique identifier for a labeling job. You can use this to refer to a specific labeling job.

Type: String

Length Constraints: Minimum length of 1.

Pattern: .+

Required: Yes

**LabelCounters**

Provides information about the progress of a labeling job.

Type: LabelCountersForWorkteam (p. 1407) object

Required: No

**LabelingJobName**

The name of the labeling job that the work team is assigned to.

Type: String


Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]+$

Required: No

**NumberOfHumanWorkersPerDataObject**

The configured number of workers per data object.

Type: Integer


Required: No

**WorkRequesterAccountId**

Type: String

Pattern: ^\d+$

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobInputConfig
Service: Amazon SageMaker Service

Input configuration information for a labeling job.

Contents

DataAttributes

Attributes of the data specified by the customer.
Type: LabelingJobDataAttributes (p. 1410) object

Required: No

DataSource

The location of the input data.
Type: LabelingJobDataSource (p. 1411) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobOutput
Service: Amazon SageMaker Service

Specifies the location of the output produced by the labeling job.

Contents

FinalActiveLearningModelArn

The Amazon Resource Name (ARN) for the most recent Amazon SageMaker model trained as part of automated data labeling.

Type: String


Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*

Required: No

OutputDatasetS3Uri

The Amazon S3 bucket location of the manifest file for labeled data.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/?(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobOutputConfig
Service: Amazon SageMaker Service

Output configuration information for a labeling job.

Contents

KmsKeyId

The AWS Key Management Service ID of the key used to encrypt the output data, if any.

If you use a KMS key ID or an alias of your master key, the Amazon SageMaker execution role must include permissions to call `kms:Encrypt`. If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. Amazon SageMaker uses server-side encryption with KMS-managed keys for `LabelingJobOutputConfig`. If you use a bucket policy with an `s3:PutObject` permission that only allows objects with server-side encryption, set the condition key of `s3:x-amz-server-side-encryption` to "aws:kms". For more information, see KMS-Managed Encryption Keys in the Amazon Simple Storage Service Developer Guide.

The KMS key policy must grant permission to the IAM role that you specify in your `CreateLabelingJob` request. For more information, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

S3OutputPath

The Amazon S3 location to write output data.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*$)

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobResourceConfig
Service: Amazon SageMaker Service

Provides configuration information for labeling jobs.

Contents

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the training job. The VolumeKmsKeyId can be any of the following formats:

- // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"
- // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobS3DataSource
Service: Amazon SageMaker Service

The Amazon S3 location of the input data objects.

Contents

ManifestS3Uri

The Amazon S3 location of the manifest file that describes the input data objects.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3):/\/([^/]+)/(\^[^/]+)/(.* )$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobStoppingConditions
Service: Amazon SageMaker Service

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.

Note
Labeling jobs fail after 30 days with an appropriate client error message.

Contents

MaxHumanLabeledObjectCount
The maximum number of objects that can be labeled by human workers.

Type: Integer
Valid Range: Minimum value of 1.
Required: No

MaxPercentageOfInputDatasetLabeled
The maximum number of input data objects that should be labeled.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobSummary
Service: Amazon SageMaker Service

Provides summary information about a labeling job.

Contents

AnnotationConsolidationLambdaArn
The Amazon Resource Name (ARN) of the Lambda function used to consolidate the annotations from individual workers into a label for a data object. For more information, see Annotation Consolidation.

Type: String
Length Constraints: Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-\_\.]\+(?:\$LATEST|[a-zA-Z0-9-\_\.]+)?
Required: No

CreationTime
The date and time that the job was created (timestamp).

Type: Timestamp
Required: Yes

FailureReason
If the LabelingJobStatus field is Failed, this field contains a description of the error.

Type: String
Length Constraints: Maximum length of 1024.
Required: No

InputConfig
Input configuration for the labeling job.

Type: LabelingJobInputConfig (p. 1414) object
Required: No

LabelCounters
Counts showing the progress of the labeling job.

Type: LabelCounters (p. 1405) object
Required: Yes

LabelingJobArn
The Amazon Resource Name (ARN) assigned to the labeling job when it was created.

Type: String
Length Constraints: Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-\*:][0-9]{12}:labeling-job/.*
Required: Yes

**LabelingJobName**

The name of the labeling job.

Type: String


Pattern: ^[^a-zA-Z0-9][^a-zA-Z0-9]*[^a-zA-Z0-9]$

Required: Yes

**LabelingJobOutput**

The location of the output produced by the labeling job.

Type: LabelingJobOutput (p. 1415) object

Required: No

**LabelingJobStatus**

The current status of the labeling job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**LastModifiedTime**

The date and time that the job was last modified (timestamp).

Type: Timestamp

Required: Yes

**PreHumanTaskLambdaArn**

The Amazon Resource Name (ARN) of a Lambda function. The function is run before each data object is sent to a worker.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:lambda:[a-z]{2}[-][a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-\-\._]*\:{\$LATEST|[a-zA-Z0-9-\-\._]*}\+$

Required: Yes

**WorkteamArn**

The Amazon Resource Name (ARN) of the work team assigned to the job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MemberDefinition
Service: Amazon SageMaker Service

Defines the Amazon Cognito user group that is part of a work team.

Contents

CognitoMemberDefinition

The Amazon Cognito user group that is part of the work team.

Type: CognitoMemberDefinition (p. 1317) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MetricData
Service: Amazon SageMaker Service

The name, value, and date and time of a metric that was emitted to Amazon CloudWatch.

Contents

MetricName
The name of the metric.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: .+
Required: No

Timestamp
The date and time that the algorithm emitted the metric.
Type: Timestamp
Required: No

Value
The value of the metric.
Type: Float
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MetricDefinition
Service: Amazon SageMaker Service

Specifies a metric that the training algorithm writes to stderr or stdout. Amazon SageMaker hyperparameter tuning captures all defined metrics. You specify one metric that a hyperparameter tuning job uses as its objective metric to choose the best training job.

Contents

Name

The name of the metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: Yes

Regex

A regular expression that searches the output of a training job and gets the value of the metric. For more information about using regular expressions to define metrics, see Defining Objective Metrics.

Type: String


Pattern: .+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelArtifacts
Service: Amazon SageMaker Service

Provides information about the location that is configured for storing model artifacts.

Contents

S3ModelArtifacts

The path of the S3 object that contains the model artifacts. For example, s3://bucket-name/keynameprefix/model.tar.gz.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageContainerDefinition
Service: Amazon SageMaker Service

Describes the Docker container for the model package.

Contents

ContainerHostname

The DNS host name for the Docker container.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored.

If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both registry/repository[:tag] and registry/repository[@digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: \S+

Required: Yes

ImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: \[Ss\][Hh][Aa]256:\[0-9a-fA-F\]{64}$

Required: No

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://(\[^/\]+)*/\([^/\]+\)/?\.*$

Required: No

ProductId

The AWS Marketplace product ID of the model package.
Type: String

Length Constraints: Maximum length of 256.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageStatusDetails
Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the model package.

Contents

ImageScanStatuses

The status of the scan of the Docker image container for the model package.

Type: Array of ModelPackageStatusItem (p. 1430) objects

Required: No

ValidationStatuses

The validation status of the model package.

Type: Array of ModelPackageStatusItem (p. 1430) objects

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageStatusItem
Service: Amazon SageMaker Service

Represents the overall status of a model package.

Contents

FailureReason
if the overall status is Failed, the reason for the failure.

Type: String
Required: No

Name
The name of the model package for which the overall status is being reported.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*#
Required: Yes

Status
The current status.

Type: String
Valid Values: NotStarted | InProgress | Completed | Failed
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageSummary
Service: Amazon SageMaker Service

Provides summary information about a model package.

Contents

CreationTime
A timestamp that shows when the model package was created.
Type: Timestamp
Required: Yes

ModelPackageArn
The Amazon Resource Name (ARN) of the model package.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*
Required: Yes

ModelPackageDescription
A brief description of the model package.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*
Required: No

ModelPackageName
The name of the model package.
Type: String
Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]*$
Required: Yes

ModelPackageStatus
The overall status of the model package.
Type: String
Valid Values: Pending | InProgress | Completed | Failed | Deleting
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
ModelPackageValidationProfile
Service: Amazon SageMaker Service

Contains data, such as the inputs and targeted instance types that are used in the process of validating the model package.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

Contents

ProfileName

The name of the profile for the model package.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

TransformJobDefinition

The TransformJobDefinition object that describes the transform job used for the validation of the model package.

Type: TransformJobDefinition (p. 1538) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageValidationSpecification
Service: Amazon SageMaker Service

Specifies batch transform jobs that Amazon SageMaker runs to validate your model package.

Contents

ValidationProfiles

An array of ModelPackageValidationProfile objects, each of which specifies a batch transform job that Amazon SageMaker runs to validate your model package.

Type: Array of ModelPackageValidationProfile (p. 1433) objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole

The IAM roles to be used for the validation of the model package.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\/]+$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelSummary
Service: Amazon SageMaker Service

Provides summary information about a model.

Contents

CreationTime

A timestamp that indicates when the model was created.

Type: Timestamp

Required: Yes

ModelArn

The Amazon Resource Name (ARN) of the model.

Type: String


Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model/.*

Required: Yes

ModelName

The name of the model that you want a summary for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**MonitoringAppSpecification**

Service: Amazon SageMaker Service

Container image configuration object for the monitoring job.

**Contents**

**ContainerArguments**

An array of arguments for the container used to run the monitoring job.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 50 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**ContainerEntrypoint**

Specifies the entrypoint for a container used to run the monitoring job.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 100 items.

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**ImageUri**

The container image to be run by the monitoring job.

Type: String

Length Constraints: Maximum length of 255.

Pattern: . *

Required: Yes

**PostAnalyticsProcessorSourceUri**

An Amazon S3 URI to a script that is called after analysis has been performed. Applicable only for the built-in (first party) containers.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)(/=)?(.*)$

Required: No

**RecordPreprocessorSourceUri**

An Amazon S3 URI to a script that is called per row prior to running analysis. It can base64 decode the payload and convert it into a flatted json so that the built-in container can use the converted data. Applicable only for the built-in (first party) containers.
Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^^(https|s3)://([\^/]+)?(\.*)$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringBaselineConfig
Service: Amazon SageMaker Service

Configuration for monitoring constraints and monitoring statistics. These baseline resources are compared against the results of the current job from the series of jobs scheduled to collect data periodically.

Contents

ConstraintsResource

The baseline constraint file in Amazon S3 that the current monitoring job should validated against.

Type: MonitoringConstraintsResource (p. 1441) object

Required: No

StatisticsResource

The baseline statistics file in Amazon S3 that the current monitoring job should be validated against.

Type: MonitoringStatisticsResource (p. 1454) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringClusterConfig
Service: Amazon SageMaker Service

Configuration for the cluster used to run model monitoring jobs.

Contents

InstanceCount

The number of ML compute instances to use in the model monitoring job. For distributed processing jobs, specify a value greater than 1. The default value is 1.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: Yes

InstanceType

The ML compute instance type for the processing job.

Type: String

Valid Values:
- ml.t3.medium
- ml.t3.large
- ml.t3.xlarge
- ml.t3.2xlarge
- ml.m4.16xlarge
- ml.m4.xlarge
- ml.m4.2xlarge
- ml.m4.4xlarge
- ml.m4.10xlarge
- ml.m4.8xlarge
- ml.p2.xlarge
- ml.p2.8xlarge
- ml.p2.16xlarge
- ml.p3.2xlarge
- ml.p3.8xlarge
- ml.p3.16xlarge
- ml.c4.xlarge
- ml.c4.2xlarge
- ml.c4.4xlarge
- ml.c4.8xlarge
- ml.c4.16xlarge
- ml.c5.xlarge
- ml.c5.2xlarge
- ml.c5.med
- ml.c5.large
- ml.c5.xlarge
- ml.c5.18xlarge
- ml.m5.xlarge
- ml.m5.2xlarge
- ml.m5.4xlarge
- ml.m5.12xlarge
- ml.m5.24xlarge
- ml.r5.xlarge
- ml.r5.2xlarge
- ml.r5.4xlarge
- ml.r5.8xlarge
- ml.r5.16xlarge
- ml.r5.24xlarge

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the model monitoring job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

VolumeSizeInGB

The size of the ML storage volume, in gigabytes, that you want to provision. You must specify sufficient ML storage for your scenario.

Type: Integer


Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
MonitoringConstraintsResource
Service: Amazon SageMaker Service

The constraints resource for a monitoring job.

Contents

S3Uri

The Amazon S3 URI for the constraints resource.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^((https|s3)://([^/]+)/?[^/]+)*$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringExecutionSummary
Service: Amazon SageMaker Service

Summary of information about the last monitoring job to run.

Contents

CreationTime
The time at which the monitoring job was created.
Type: Timestamp
Required: Yes

EndpointName
The name of the endpoint used to run the monitoring job.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: No

FailureReason
Contains the reason a monitoring job failed, if it failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

LastModifiedTime
A timestamp that indicates the last time the monitoring job was modified.
Type: Timestamp
Required: Yes

MonitoringExecutionStatus
The status of the monitoring job.
Type: String
Valid Values: Pending | Completed | CompletedWithViolations | InProgress | Failed | Stopping | Stopped
Required: Yes

MonitoringScheduleName
The name of the monitoring schedule.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
**Required: Yes**

**ProcessingJobArn**

The Amazon Resource Name (ARN) of the monitoring job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:processing-job/.*`

Required: No

**ScheduledTime**

The time the monitoring job was scheduled.

Type: Timestamp

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringInput
Service: Amazon SageMaker Service

The inputs for a monitoring job.

Contents

EndpointInput

The endpoint for a monitoring job.

Type: EndpointInput (p. 1342) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringJobDefinition
Service: Amazon SageMaker Service

Defines the monitoring job.

Contents

BaselineConfig
Baseline configuration used to validate that the data conforms to the specified constraints and statistics
Type: MonitoringBaselineConfig (p. 1438) object
Required: No

Environment
Sets the environment variables in the Docker container.
Type: String to string map
Key Length Constraints: Maximum length of 256.
Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*
Value Length Constraints: Maximum length of 256.
Value Pattern: [\S\s]*
Required: No

MonitoringAppSpecification
Configures the monitoring job to run a specified Docker container image.
Type: MonitoringAppSpecification (p. 1436) object
Required: Yes

MonitoringInputs
The array of inputs for the monitoring job. Currently we support monitoring an Amazon SageMaker Endpoint.
Type: Array of MonitoringInput (p. 1444) objects
Array Members: Fixed number of 1 item.
Required: Yes

MonitoringOutputConfig
The array of outputs from the monitoring job to be uploaded to Amazon Simple Storage Service (Amazon S3).
Type: MonitoringOutputConfig (p. 1448) object
Required: Yes

MonitoringResources
Identifies the resources, ML compute instances, and ML storage volumes to deploy for a monitoring job. In distributed processing, you specify more than one instance.
Type: `MonitoringResources (p. 1449)` object

Required: Yes

**NetworkConfig**

Specifies networking options for an monitoring job.

Type: `NetworkConfig (p. 1457)` object

Required: No

**RoleArn**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\[/]+$`

Required: Yes

**StoppingCondition**

Specifies a time limit for how long the monitoring job is allowed to run.

Type: `MonitoringStoppingCondition (p. 1455)` object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringOutput
Service: Amazon SageMaker Service

The output object for a monitoring job.

Contents

S3Output

The Amazon S3 storage location where the results of a monitoring job are saved.

Type: MonitoringS3Output (p. 1450) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringOutputConfig

Service: Amazon SageMaker Service

The output configuration for monitoring jobs.

Contents

KmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: . *

Required: No

MonitoringOutputs

Monitoring outputs for monitoring jobs. This is where the output of the periodic monitoring jobs is uploaded.

Type: Array of MonitoringOutput (p. 1447) objects

Array Members: Fixed number of 1 item.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringResources
Service: Amazon SageMaker Service

Identifies the resources to deploy for a monitoring job.

Contents

ClusterConfig

The configuration for the cluster resources used to run the processing job.

Type: MonitoringClusterConfig (p. 1439) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringS3Output
Service: Amazon SageMaker Service

Information about where and how you want to store the results of a monitoring job.

Contents

LocalPath

The local path to the Amazon S3 storage location where Amazon SageMaker saves the results of a monitoring job. LocalPath is an absolute path for the output data.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

S3UploadMode

Whether to upload the results of the monitoring job continuously or after the job completes.

Type: String

Valid Values: Continuous | EndOfJob

Required: No

S3Uri

A URI that identifies the Amazon S3 storage location where Amazon SageMaker saves the results of a monitoring job.

Type: String

Length Constraints: Maximum length of 512.

Pattern: ^(https|s3)://([^/]+)/(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringScheduleConfig
Service: Amazon SageMaker Service

Configures the monitoring schedule and defines the monitoring job.

Contents

MonitoringJobDefinition

Defines the monitoring job.

Type: MonitoringJobDefinition (p. 1445) object

Required: Yes

ScheduleConfig

Configures the monitoring schedule.

Type: ScheduleConfig (p. 1503) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringScheduleSummary
Service: Amazon SageMaker Service

Summarizes the monitoring schedule.

Contents

CreationTime
The creation time of the monitoring schedule.
Type: Timestamp
Required: Yes

EndpointName
The name of the endpoint using the monitoring schedule.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
Required: No

LastModifiedTime
The last time the monitoring schedule was modified.
Type: Timestamp
Required: Yes

MonitoringScheduleArn
The Amazon Resource Name (ARN) of the monitoring schedule.
Type: String
Length Constraints: Maximum length of 256.
Pattern: .*
Required: Yes

MonitoringScheduleName
The name of the monitoring schedule.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*#
Required: Yes

MonitoringScheduleStatus
The status of the monitoring schedule.
Type: String
Valid Values: Pending | Failed | Scheduled | Stopped

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringStatisticsResource
Service: Amazon SageMaker Service

The statistics resource for a monitoring job.

Contents

S3Uri

The Amazon S3 URI for the statistics resource.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*$)

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MonitoringStoppingCondition
Service: Amazon SageMaker Service

A time limit for how long the monitoring job is allowed to run before stopping.

Contents

MaxRuntimeInSeconds

The maximum runtime allowed in seconds.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 86400.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NestedFilters
Service: Amazon SageMaker Service

Defines a list of NestedFilters objects. To satisfy the conditions specified in the NestedFilters call, a resource must satisfy the conditions of all of the filters.

For example, you could define a NestedFilters using the training job's InputDataConfig property to filter on Channel objects.

A NestedFilters object contains multiple filters. For example, to find all training jobs whose name contains train and that have cat/data in their S3Uri (specified in InputDataConfig), you need to create a NestedFilters object that specifies the InputDataConfig property with the following Filter objects:

- '{Name: "InputDataConfig.ChannelName", "Operator": "EQUALS", "Value": "train"}',
- '{Name: "InputDataConfig.DataSource.S3DataSource.S3Uri", "Operator": "CONTAINS", "Value": "cat/data"}'

Contents

Filters

A list of filters. Each filter acts on a property. Filters must contain at least one Filters value. For example, a NestedFilters call might include a filter on the PropertyName parameter of the InputDataConfig property: InputDataConfig.DataSource.S3DataSource.S3Uri.

Type: Array of Filter (p. 1354) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: Yes

NestedPropertyName

The name of the property to use in the nested filters. The value must match a listed property name, such as InputDataConfig.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NetworkConfig
Service: Amazon SageMaker Service

Networking options for a job, such as network traffic encryption between containers, whether to allow inbound and outbound network calls to and from containers, and the VPC subnets and security groups to use for VPC-enabled jobs.

Contents

EnableNetworkIsolation

Whether to allow inbound and outbound network calls to and from the containers used for the processing job.

Type: Boolean

Required: No

VpcConfig

Specifies a VPC that your training jobs and hosted models have access to. Control access to and from your training and model containers by configuring the VPC. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceLifecycleConfigSummary
Service: Amazon SageMaker Service

Provides a summary of a notebook instance lifecycle configuration.

Contents

CreationTime
A timestamp that tells when the lifecycle configuration was created.
Type: Timestamp
Required: No

LastModifiedTime
A timestamp that tells when the lifecycle configuration was last modified.
Type: Timestamp
Required: No

NotebookInstanceLifecycleConfigArn
The Amazon Resource Name (ARN) of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 256.
Required: Yes

NotebookInstanceLifecycleConfigName
The name of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9\-\_0-9]*[a-zA-Z0-9\-\_0-9]+$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceLifecycleHook
Service: Amazon SageMaker Service

Contains the notebook instance lifecycle configuration script.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the `$PATH` environment variable that is available to both scripts is `/sbin:/bin:/usr/sbin:/usr/bin`.


Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Contents

Content

A base64-encoded string that contains a shell script for a notebook instance lifecycle configuration.

Type: String


Pattern: `\S\S+`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceSummary
Service: Amazon SageMaker Service

Provides summary information for an Amazon SageMaker notebook instance.

Contents

AdditionalCodeRepositories

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings
Array Members: Maximum number of 3 items.
Pattern: ^https://(\^[^/]+)/(.*$)|^[a-zA-Z0-9]+(-*[a-zA-Z0-9]*)*$
Required: No

CreationTime

A timestamp that shows when the notebook instance was created.
Type: Timestamp
Required: No

DefaultCodeRepository

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String
Pattern: ^https://(\^[^/]+)/(.*$)|^[a-zA-Z0-9]+(-*[a-zA-Z0-9]*)*$
Required: No

InstanceType

The type of ML compute instance that the notebook instance is running on.
Type: String
Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge |

| Required | No |

**LastModifiedTime**

A timestamp that shows when the notebook instance was last modified.

<table>
<thead>
<tr>
<th>Type</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>No</td>
</tr>
</tbody>
</table>

**NotebookInstanceArn**

The Amazon Resource Name (ARN) of the notebook instance.

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length Constraints</td>
<td>Maximum length of 256</td>
</tr>
<tr>
<td>Required</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NotebookInstanceLifecycleConfigName**

The name of a notebook instance lifecycle configuration associated with this notebook instance.

For information about notebook instance lifestyle configurations, see [Step 2.1: (Optional) Customize a Notebook Instance](#).

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length Constraints</td>
<td>Maximum length of 63</td>
</tr>
<tr>
<td>Pattern</td>
<td>^<a href="-*%5Ba-zA-Z0-9%5D*">a-zA-Z0-9</a>*</td>
</tr>
<tr>
<td>Required</td>
<td>No</td>
</tr>
</tbody>
</table>

**NotebookInstanceName**

The name of the notebook instance that you want a summary for.

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length Constraints</td>
<td>Maximum length of 63</td>
</tr>
<tr>
<td>Pattern</td>
<td>^<a href="-*%5Ba-zA-Z0-9%5D*">a-zA-Z0-9</a>*</td>
</tr>
<tr>
<td>Required</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**NotebookInstanceStatus**

The status of the notebook instance.

<table>
<thead>
<tr>
<th>Valid Values</th>
<th>Pending</th>
<th>InService</th>
<th>Stopping</th>
<th>Stopped</th>
<th>Failed</th>
<th>Deleting</th>
<th>Updating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Url**

The URL that you use to connect to the Jupyter instance running in your notebook instance.

| Type | String |
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotificationConfiguration
Service: Amazon SageMaker Service

Configures SNS notifications of available or expiring work items for work teams.

Contents

NotificationTopicArn

The ARN for the SNS topic to which notifications should be published.

Type: String

Pattern: arn:aws[a-z\-]*:sns:[a-z0-9\-]*:[0-9]{12}:[a-zA-Z0-9-_.-]*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ObjectiveStatusCounters
Service: Amazon SageMaker Service

Specifies the number of training jobs that this hyperparameter tuning job launched, categorized by the status of their objective metric. The objective metric status shows whether the final objective metric for the training job has been evaluated by the tuning job and used in the hyperparameter tuning process.

Contents

Failed

The number of training jobs whose final objective metric was not evaluated and used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Pending

The number of training jobs that are in progress and pending evaluation of their final objective metric.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Succeeded

The number of training jobs whose final objective metric was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
OutputConfig
Service: Amazon SageMaker Service

Contains information about the output location for the compiled model and the device (target) that the model runs on.

Contents

S3OutputLocation

Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, s3://bucket-name/key-name-prefix.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3):/(/[^/]+)?(/.*)$
Required: Yes

TargetDevice

Identifies the device that you want to run your model on after it has been compiled. For example: ml_c5.

Type: String
Valid Values: lambda | ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | ml_inf1 | jetson_tx1 | jetson_tx2 | jetsonNano | rasp3b | deeplens | rk3399 | rk3288 | aisage | sbe_c | qcs605 | qcs603
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
OutputDataConfig
Service: Amazon SageMaker Service

Provides information about how to store model training results (model artifacts).

Contents

KmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The KmsKeyId can be any of the following formats:

- // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"
- // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
- // KMS Key Alias
  "alias/ExampleAlias"
- // Amazon Resource Name (ARN) of a KMS Key Alias

If you use a KMS key ID or an alias of your master key, the Amazon SageMaker execution role must include permissions to call kms:Encrypt. If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. Amazon SageMaker uses server-side encryption with KMS-managed keys for OutputDataConfig. If you use a bucket policy with an s3:PutObject permission that only allows objects with server-side encryption, set the condition key of s3:x-amz-server-side-encryption to "aws:kms". For more information, see KMS-Managed Encryption Keys in the Amazon Simple Storage Service Developer Guide.

The KMS key policy must grant permission to the IAM role that you specify in your CreateTrainingJob, CreateTransformJob, or CreateHyperParameterTuningJob requests. For more information, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: No

S3OutputPath

Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, s3://bucket-name/key-name-prefix.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3):/([^/]+)/?(.*)$
Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ParameterRange
Service: Amazon SageMaker Service

Defines the possible values for categorical, continuous, and integer hyperparameters to be used by an algorithm.

Contents

CategoricalParameterRangeSpecification

A CategoricalParameterRangeSpecification object that defines the possible values for a categorical hyperparameter.

Type: CategoricalParameterRangeSpecification (p. 1309) object

Required: No

ContinuousParameterRangeSpecification

A ContinuousParameterRangeSpecification object that defines the possible values for a continuous hyperparameter.

Type: ContinuousParameterRangeSpecification (p. 1325) object

Required: No

IntegerParameterRangeSpecification

A IntegerParameterRangeSpecification object that defines the possible values for an integer hyperparameter.

Type: IntegerParameterRangeSpecification (p. 1402) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ParameterRanges
Service: Amazon SageMaker Service

Specifies ranges of integer, continuous, and categorical hyperparameters that a hyperparameter tuning job searches. The hyperparameter tuning job launches training jobs with hyperparameter values within these ranges to find the combination of values that result in the training job with the best performance as measured by the objective metric of the hyperparameter tuning job.

Note
You can specify a maximum of 20 hyperparameters that a hyperparameter tuning job can search over. Every possible value of a categorical parameter range counts against this limit.

Contents

CategoricalParameterRanges
The array of CategoricalParameterRange (p. 1308) objects that specify ranges of categorical hyperparameters that a hyperparameter tuning job searches.

Type: Array of CategoricalParameterRange (p. 1308) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

ContinuousParameterRanges
The array of ContinuousParameterRange (p. 1323) objects that specify ranges of continuous hyperparameters that a hyperparameter tuning job searches.

Type: Array of ContinuousParameterRange (p. 1323) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

IntegerParameterRanges
The array of IntegerParameterRange (p. 1400) objects that specify ranges of integer hyperparameters that a hyperparameter tuning job searches.

Type: Array of IntegerParameterRange (p. 1400) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Parent
Service: Amazon SageMaker Service

The trial that a trial component is associated with and the experiment the trial is part of. A component might not be associated with a trial. A component can be associated with multiple trials.

Contents

ExperimentName

The name of the experiment.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

TrialName

The name of the trial.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ParentHyperParameterTuningJob
Service: Amazon SageMaker Service

A previously completed or stopped hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Contents

HyperParameterTuningJobName

The name of the hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Type: String


Pattern: \^[a-zA-Z0-9](-*[a-zA-Z0-9])*\n
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingClusterConfig
Service: Amazon SageMaker Service

Configuration for the cluster used to run a processing job.

Contents

InstanceCount

The number of ML compute instances to use in the processing job. For distributed processing jobs, specify a value greater than 1. The default value is 1.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: Yes

InstanceType

The ML compute instance type for the processing job.

Type: String

Valid Values: ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.r5.large | ml.r5.xlarge | ml.r5.2xlarge | ml.r5.4xlarge | ml.r5.8xlarge | ml.r5.12xlarge | ml.r5.16xlarge | ml.r5.24xlarge

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the processing job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

VolumeSizeInGB

The size of the ML storage volume in gigabytes that you want to provision. You must specify sufficient ML storage for your scenario.

Type: Integer


Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
ProcessingInput
Service: Amazon SageMaker Service

The inputs for a processing job.

Contents

InputName

The name of the inputs for the processing job.

Type: String

Required: Yes

S3Input

The S3 inputs for the processing job.

Type: ProcessingS3Input (p. 1480) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ProcessingJobSummary**  
Service: Amazon SageMaker Service

Summary of information about a processing job.

**Contents**

**CreationTime**

The time at which the processing job was created.

Type: Timestamp

Required: Yes

**ExitMessage**

An optional string, up to one KB in size, that contains metadata from the processing container when the processing job exits.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: \S\s*

Required: No

**FailureReason**

A string, up to one KB in size, that contains the reason a processing job failed, if it failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

**LastModifiedTime**

A timestamp that indicates the last time the processing job was modified.

Type: Timestamp

Required: No

**ProcessingEndTime**

The time at which the processing job completed.

Type: Timestamp

Required: No

**ProcessingJobArn**

The Amazon Resource Name (ARN) of the processing job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:processing-job/.*

Required: Yes
**ProcessingJobName**

The name of the processing job.

Type: String


Pattern: `^[a-zA-Z0-9](\*-[a-zA-Z0-9])*`

Required: Yes

**ProcessingJobStatus**

The status of the processing job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ProcessingOutput**
Service: Amazon SageMaker Service

Describes the results of a processing job.

**Contents**

**OutputName**

The name for the processing job output.

Type: String

Required: Yes

**S3Output**

Configuration for processing job outputs in Amazon S3.

Type: ProcessingS3Output (p. 1482) object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingOutputConfig
Service: Amazon SageMaker Service

The output configuration for the processing job.

Contents

KmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the processing job output. KmsKeyId can be an ID of a KMS key, ARN of a KMS key, alias of a KMS key, or alias of a KMS key. The KmsKeyId is applied to all outputs.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

Outputs

Output configuration information for a processing job.

Type: Array of ProcessingOutput (p. 1477) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingResources
Service: Amazon SageMaker Service

Identifies the resources, ML compute instances, and ML storage volumes to deploy for a processing job. In distributed training, you specify more than one instance.

Contents

ClusterConfig

The configuration for the resources in a cluster used to run the processing job.

Type: ProcessingClusterConfig (p. 1472) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingS3Input
Service: Amazon SageMaker Service

Information about where and how you want to obtain the inputs for an processing job.

Contents

LocalPath

The local path to the Amazon S3 bucket where you want Amazon SageMaker to download the inputs to run a processing job. LocalPath is an absolute path to the input data.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

S3CompressionType

Whether to use Gzip compression for Amazon S3 storage.

Type: String

Valid Values: None | Gzip

Required: No

S3DataDistributionType

Whether the data stored in Amazon S3 is FullyReplicated or ShardedByS3Key.

Type: String

Valid Values: FullyReplicated | ShardedByS3Key

Required: No

S3DataType

Whether you use an S3Prefix or a ManifestFile for the data type. If you choose S3Prefix, S3Uri identifies a key name prefix. Amazon SageMaker uses all objects with the specified key name prefix for the processing job. If you choose ManifestFile, S3Uri identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for the processing job.

Type: String

Valid Values: ManifestFile | S3Prefix

Required: Yes

S3InputMode

Wether to use File or Pipe input mode. In File mode, Amazon SageMaker copies the data from the input source onto the local Amazon Elastic Block Store (Amazon EBS) volumes before starting your training algorithm. This is the most commonly used input mode. In Pipe mode, Amazon SageMaker streams input data from the source directly to your algorithm without using the EBS volume.

Type: String
Valid Values: Pipe | File
Required: Yes

**S3Uri**

The URI for the Amazon S3 storage where you want Amazon SageMaker to download the artifacts needed to run a processing job.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/(\.|\?)[\^+].*$`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingS3Output
Service: Amazon SageMaker Service

Information about where and how you want to store the results of an processing job.

Contents

LocalPath

The local path to the Amazon S3 bucket where you want Amazon SageMaker to save the results of an processing job. LocalPath is an absolute path to the input data.

Type: String

Length Constraints: Maximum length of 256.

Pattern: .*

Required: Yes

S3UploadMode

Whether to upload the results of the processing job continuously or after the job completes.

Type: String

Valid Values: Continuous | EndOfJob

Required: Yes

S3Uri

A URI that identifies the Amazon S3 bucket where you want Amazon SageMaker to save the results of a processing job.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/?([./]?[^$]+)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProcessingStoppingCondition
Service: Amazon SageMaker Service

Specifies a time limit for how long the processing job is allowed to run.

Contents

MaxRuntimeInSeconds

Specifies the maximum runtime in seconds.

Type: Integer


Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ProductionVariant**

Service: Amazon SageMaker Service

Identifies a model that you want to host and the resources to deploy for hosting it. If you are deploying multiple models, tell Amazon SageMaker how to distribute traffic among the models by specifying variant weights.

**Contents**

**AcceleratorType**

The size of the Elastic Inference (EI) instance to use for the production variant. EI instances provide on-demand GPU computing for inference. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: String

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge | ml.eia2.medium | ml.eia2.large | ml.eia2.xlarge

Required: No

**InitialInstanceCount**

Number of instances to launch initially.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

**InitialVariantWeight**

Determines initial traffic distribution among all of the models that you specify in the endpoint configuration. The traffic to a production variant is determined by the ratio of the VariantWeight to the sum of all VariantWeight values across all ProductionVariants. If unspecified, it defaults to 1.0.

Type: Float

Valid Range: Minimum value of 0.

Required: No

**InstanceType**

The ML compute instance type.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.m5d.large | ml.m5d.xlarge | ml.m5d.2xlarge | ml.m5d.4xlarge | ml.m5d.12xlarge | ml.m5d.24xlarge | ml.c4.large | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.large | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge | ml.c5d.18xlarge | ml.g4dn.xlarge | ml.g4dn.2xlarge | ml.g4dn.4xlarge

1484
ModelName

The name of the model that you want to host. This is the name that you specified when creating the model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

VariantName

The name of the production variant.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProductionVariantSummary
Service: Amazon SageMaker Service

Describes weight and capacities for a production variant associated with an endpoint. If you sent a request to the UpdateEndpointWeightsAndCapacities API and the endpoint status is Updating, you get different desired and current values.

Contents

CurrentInstanceCount
  The number of instances associated with the variant.
  Type: Integer
  Valid Range: Minimum value of 1.
  Required: No

CurrentWeight
  The weight associated with the variant.
  Type: Float
  Valid Range: Minimum value of 0.
  Required: No

DeployedImages
  An array of DeployedImage objects that specify the Amazon EC2 Container Registry paths of the inference images deployed on instances of this ProductionVariant.
  Type: Array of DeployedImage (p. 1337) objects
  Required: No

DesiredInstanceCount
  The number of instances requested in the UpdateEndpointWeightsAndCapacities request.
  Type: Integer
  Valid Range: Minimum value of 1.
  Required: No

DesiredWeight
  The requested weight, as specified in the UpdateEndpointWeightsAndCapacities request.
  Type: Float
  Valid Range: Minimum value of 0.
  Required: No

VariantName
  The name of the variant.
  Type: String
  Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**PropertyNameQuery**  
Service: Amazon SageMaker Service  

Part of the `SuggestionQuery` type. Specifies a hint for retrieving property names that begin with the specified text.

**Contents**

**PropertyNameHint**

Text that begins a property's name.  
Type: String  
Length Constraints: Minimum length of 0. Maximum length of 100.  
Pattern: .*  
Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
PropertyNameSuggestion
Service: Amazon SageMaker Service

A property name returned from a GetSearchSuggestions call that specifies a value in the PropertyNameQuery field.

Contents

PropertyName

A suggested property name based on what you entered in the search textbox in the Amazon SageMaker console.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: .+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
PublicWorkforceTaskPrice
Service: Amazon SageMaker Service

Defines the amount of money paid to an Amazon Mechanical Turk worker for each task performed.

Use one of the following prices for bounding box tasks. Prices are in US dollars and should be based on the complexity of the task; the longer it takes in your initial testing, the more you should offer.

- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for image classification, text classification, and custom tasks. Prices are in US dollars.

- 0.012
- 0.024
- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for semantic segmentation tasks. Prices are in US dollars.

- 0.840
- 0.960
- 1.080
• 1.200

Use one of the following prices for Textract AnalyzeDocument Important Form Key Amazon Augmented AI review tasks. Prices are in US dollars.

• 2.400
• 2.280
• 2.160
• 2.040
• 1.920
• 1.800
• 1.680
• 1.560
• 1.440
• 1.320
• 1.200
• 1.080
• 0.960
• 0.840
• 0.720
• 0.600
• 0.480
• 0.360
• 0.240
• 0.120
• 0.072
• 0.060
• 0.048
• 0.036
• 0.024
• 0.012

Use one of the following prices for Rekognition DetectModerationLabels Amazon Augmented AI review tasks. Prices are in US dollars.

• 1.200
• 1.080
• 0.960
• 0.840
• 0.720
• 0.600
• 0.480
• 0.360
• 0.240
• 0.120
• 0.072
• 0.060
Use one of the following prices for Amazon Augmented AI custom human review tasks. Prices are in US dollars.

- 1.200
- 1.080
- 0.960
- 0.840
- 0.720
- 0.600
- 0.480
- 0.360
- 0.240
- 0.120
- 0.072
- 0.060
- 0.048
- 0.036
- 0.024
- 0.012

**Contents**

**AmountInUsd**

Defines the amount of money paid to an Amazon Mechanical Turk worker in United States dollars.

Type: USD (p. 1571) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
RenderableTask
Service: Amazon SageMaker Service
Contains input values for a task.

Contents

Input

A JSON object that contains values for the variables defined in the template. It is made available to the template under the substitution variable task.input. For example, if you define a variable task.input.text in your template, you can supply the variable in the JSON object as "text": "sample text".

Type: String


Pattern: [\S\s]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
RenderingError
Service: Amazon SageMaker Service

A description of an error that occurred while rendering the template.

Contents

Code
A unique identifier for a specific class of errors.

Type: String
Required: Yes

Message
A human-readable message describing the error.

Type: String
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ResolvedAttributes
Service: Amazon SageMaker Service
The resolved attributes.

Contents

AutoMLJobObjective
Applies a metric to minimize or maximize for the job's objective.
Type: AutoMLJobObjective (p. 1300) object
Required: No

CompletionCriteria
How long a job is allowed to run, or how many candidates a job is allowed to generate.
Type: AutoMLJobCompletionCriteria (p. 1298) object
Required: No

ProblemType
The problem type.
Type: String
Valid Values: BinaryClassification | MulticlassClassification | Regression
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ResourceConfig
Service: Amazon SageMaker Service

Describes the resources, including ML compute instances and ML storage volumes, to use for model training.

Contents

InstanceCount

The number of ML compute instances to use. For distributed training, provide a value greater than 1.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType

The ML compute instance type.

Type: String

Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.p3dn.24xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge

Required: Yes

VolumeKmsKeyId

The AWS KMS key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the training job.

Note

Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a VolumeKmsKeyId when using an instance type with local storage.

For a list of instance types that support local instance storage, see Instance Store Volumes.

For more information about local instance storage encryption, see SSD Instance Store Volumes.

The VolumeKmsKeyId can be in any of the following formats:

- // KMS Key ID

  "1234abcd-12ab-34cd-56ef-1234567890ab"

- // Amazon Resource Name (ARN) of a KMS Key

  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*
VolumeSizeInGB

The size of the ML storage volume that you want to provision.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use the ML storage volume for scratch space. If you want to store the training data in the ML storage volume, choose File as the TrainingInputMode in the algorithm specification.

You must specify sufficient ML storage for your scenario.

Note
Amazon SageMaker supports only the General Purpose SSD (gp2) ML storage volume type.

Note
Certain Nitro-based instances include local storage with a fixed total size, dependent on the instance type. When using these instances for training, Amazon SageMaker mounts the local instance storage instead of Amazon EBS gp2 storage. You can't request a VolumeSizeInGB greater than the total size of the local instance storage.

For a list of instance types that support local instance storage, including the total size per instance type, see Instance Store Volumes.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ResourceLimits
Service: Amazon SageMaker Service

Specifies the maximum number of training jobs and parallel training jobs that a hyperparameter tuning job can launch.

Contents

MaxNumberOfTrainingJobs

The maximum number of training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

MaxParallelTrainingJobs

The maximum number of concurrent training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ResourceSpec
Service: Amazon SageMaker Service

The instance type and quantity.

Contents

EnvironmentArn

The Amazon Resource Name (ARN) of the environment.

Type: String

Length Constraints: Maximum length of 256.

Pattern: ^arn:aws(-[\w]+)*:sagemaker:.+:[0-9]{12}:environment/[a-z0-9]{0,62}*

Required: No

InstanceType

The instance type.

Type: String

Valid Values: system | ml.t3.micro | ml.t3.small | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.8xlarge | ml.m5.12xlarge | ml.m5.16xlarge | ml.m5.24xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.12xlarge | ml.c5.18xlarge | ml.c5.24xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.g4dn.xlarge | ml.g4dn.2xlarge | ml.g4dn.4xlarge | ml.g4dn.8xlarge | ml.g4dn.12xlarge | ml.g4dn.16xlarge

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
RetentionPolicy
Service: Amazon SageMaker Service

The retention policy.

Contents

HomeEfsFileSystem
The home Amazon Elastic File System (EFS).

Type: String

Valid Values: Retain | Delete

Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
S3DataSource
Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

AttributeNames
A list of one or more attribute names to use that are found in a specified augmented manifest file.

Type: Array of strings

Array Members: Maximum number of 16 items.

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: .+

Required: No

S3DataDistributionType
If you want Amazon SageMaker to replicate the entire dataset on each ML compute instance that is launched for model training, specify FullyReplicated.

If you want Amazon SageMaker to replicate a subset of data on each ML compute instance that is launched for model training, specify ShardedByS3Key. If there are $n$ ML compute instances launched for a training job, each instance gets approximately $1/n$ of the number of S3 objects. In this case, model training on each machine uses only the subset of training data.

Don't choose more ML compute instances for training than available S3 objects. If you do, some nodes won't get any data and you will pay for nodes that aren't getting any training data. This applies in both File and Pipe modes. Keep this in mind when developing algorithms.

In distributed training, where you use multiple ML compute EC2 instances, you might choose ShardedByS3Key. If the algorithm requires copying training data to the ML storage volume (when TrainingInputMode is set to File), this copies $1/n$ of the number of objects.

Type: String

Valid Values: FullyReplicated | ShardedByS3Key

Required: No

S3DataType
If you choose S3Prefix, S3Uri identifies a key name prefix. Amazon SageMaker uses all objects that match the specified key name prefix for model training.

If you choose ManifestFile, S3Uri identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for model training.

If you choose AugmentedManifestFile, S3Uri identifies an object that is an augmented manifest file in JSON lines format. This file contains the data you want to use for model training. AugmentedManifestFile can only be used if the Channel's input mode is Pipe.

Type: String

Valid Values: ManifestFile | S3Prefix | AugmentedManifestFile

Required: Yes
S3Uri

Depending on the value specified for the S3DataType, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: s3://bucketname/exampleprefix.
- A manifest might look like this: s3://bucketname/example.manifest

The manifest is an S3 object which is a JSON file with the following format:

The preceding JSON matches the following s3Uris:

```
[ {"prefix": "s3://customer_bucket/some/prefix/"},
  "relative/path/to/custdata-1",
  "relative/path/custdata-2",
  ...
  "relative/path/custdata-N"
]
```

The preceding JSON matches the following s3Uris:

```
s3://customer_bucket/some/prefix/relative/path/to/custdata-1
s3://customer_bucket/some/prefix/relative/path/custdata-2
...
```

The complete set of s3Uris in this manifest is the input data for the channel for this datasource. The object that each s3Uri points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://(\[^/\]+)/?\([^/]+\)/?\(\.*\)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ScheduleConfig**  
Service: Amazon SageMaker Service

Configuration details about the monitoring schedule.

**Contents**

**ScheduleExpression**

A cron expression that describes details about the monitoring schedule.

Currently the only supported cron expressions are:

- If you want to set the job to start every hour, please use the following:
  
  Hourly: cron(0 * ? * *)

- If you want to start the job daily:

  cron(0 [00-23] ? * *)

For example, the following are valid cron expressions:

- Daily at noon UTC: cron(0 12 ? * *)
- Daily at midnight UTC: cron(0 0 ? * *)

To support running every 6, 12 hours, the following are also supported:

  cron(0 [00-23]/[01-24] ? * *)

For example, the following are valid cron expressions:

- Every 12 hours, starting at 5pm UTC: cron(0 17/12 ? * *)
- Every two hours starting at midnight: cron(0 0/2 ? * *)

**Note**

- Even though the cron expression is set to start at 5PM UTC, note that there could be a delay of 0-20 minutes from the actual requested time to run the execution.
- We recommend that if you would like a daily schedule, you do not provide this parameter. Amazon SageMaker will pick a time for running every day.

**Type:** String

**Length Constraints:** Minimum length of 1. Maximum length of 256.

**Required:** Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SearchExpression

Service: Amazon SageMaker Service

A multi-expression that searches for the specified resource or resources in a search. All resource objects that satisfy the expression's condition are included in the search results. You must specify at least one subexpression, filter, or nested filter. A SearchExpression can contain up to twenty elements.

A SearchExpression contains the following components:

- A list of Filter objects. Each filter defines a simple Boolean expression comprised of a resource property name, Boolean operator, and value. A SearchExpression can include only one Contains operator.
- A list of NestedFilter objects. Each nested filter defines a list of Boolean expressions using a list of resource properties. A nested filter is satisfied if a single object in the list satisfies all Boolean expressions.
- A list of SearchExpression objects. A search expression object can be nested in a list of search expression objects.
- A Boolean operator: And or Or.

Contents

Filters

A list of filter objects.

Type: Array of Filter (p. 1354) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

NestedFilters

A list of nested filter objects.

Type: Array of NestedFilters (p. 1456) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

Operator

A Boolean operator used to evaluate the search expression. If you want every conditional statement in all lists to be satisfied for the entire search expression to be true, specify And. If only a single conditional statement needs to be true for the entire search expression to be true, specify Or. The default value is And.

Type: String

Valid Values: And | Or

Required: No

SubExpressions

A list of search expression objects.

Type: Array of SearchExpression (p. 1504) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**SearchRecord**
Service: Amazon SageMaker Service

An individual search result record that contains a single resource object.

**Contents**

**Experiment**

A summary of the properties of an experiment.

Type: Experiment (p. 1346) object

Required: No

**TrainingJob**

A TrainingJob object that is returned as part of a Search request.

Type: TrainingJob (p. 1520) object

Required: No

**Trial**

A summary of the properties of a trial.

Type: Trial (p. 1548) object

Required: No

**TrialComponent**

A summary of the properties of a trial component.

Type: TrialComponent (p. 1550) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SecondaryStatusTransition
Service: Amazon SageMaker Service

An array element of DescribeTrainingJob:SecondaryStatusTransitions (p. 1072). It provides additional
details about a status that the training job has transitioned through. A training job can be in one of
several states, for example, starting, downloading, training, or uploading. Within each state, there are
a number of intermediate states. For example, within the starting state, Amazon SageMaker could be
starting the training job or launching the ML instances. These transitional states are referred to as the
job's secondary status.

Contents

EndTime

A timestamp that shows when the training job transitioned out of this secondary status state into
another secondary status state or when the training job has ended.

Type: Timestamp
Required: No

StartTime

A timestamp that shows when the training job transitioned to the current secondary status state.

Type: Timestamp
Required: Yes

Status

Contains a secondary status information from a training job.

Status might be one of the following secondary statuses:

InProgress
• Starting - Starting the training job.
• Downloading - An optional stage for algorithms that support File training input mode. It
  indicates that data is being downloaded to the ML storage volumes.
• Training - Training is in progress.
• Uploading - Training is complete and the model artifacts are being uploaded to the S3
  location.

Completed
• Completed - The training job has completed.

Failed
• Failed - The training job has failed. The reason for the failure is returned in the
  FailureReason field of DescribeTrainingJobResponse.

Stopped
• MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed
  runtime.
• Stopped - The training job has stopped.

Stopping
• Stopping - Stopping the training job.

We no longer support the following secondary statuses:
• LaunchingMLInstances
• PreparingTrainingStack
• DownloadingTrainingImage

Type: String

Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded

Required: Yes

**StatusMessage**

A detailed description of the progress within a secondary status.

Amazon SageMaker provides secondary statuses and status messages that apply to each of them:

**Starting**

- Starting the training job.
- Launching requested ML instances.
- Insufficient capacity error from EC2 while launching instances, retrying!
- Launched instance was unhealthy, replacing it!
- Preparing the instances for training.

**Training**

- Downloading the training image.
- Training image download completed. Training in progress.

**Important**

Status messages are subject to change. Therefore, we recommend not including them in code that programmatically initiates actions. For examples, don't use status messages in if statements.

To have an overview of your training job's progress, view **TrainingJobStatus** and **SecondaryStatus** in **DescribeTrainingJob** (p. 1066), and **StatusMessage** together. For example, at the start of a training job, you might see the following:

- **TrainingJobStatus** - InProgress
- **SecondaryStatus** - Training
- **StatusMessage** - Downloading the training image

Type: String

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SharingSettings
Service: Amazon SageMaker Service

The sharing settings.

Contents

NotebookOutputOption

The notebook output option.

Type: String

Valid Values: Allowed | Disabled

Required: No

S3KmsKeyId

The AWS Key Management Service encryption key ID.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

S3OutputPath

The Amazon S3 output path.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(^/[^/]+)?(.*)$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**ShuffleConfig**
Service: Amazon SageMaker Service

A configuration for a shuffle option for input data in a channel. If you use `S3Prefix` for `S3DataType`, the results of the S3 key prefix matches are shuffled. If you use `ManifestFile`, the order of the S3 object references in the `ManifestFile` is shuffled. If you use `AugmentedManifestFile`, the order of the JSON lines in the `AugmentedManifestFile` is shuffled. The shuffling order is determined using the `Seed` value.

For Pipe input mode, when `ShuffleConfig` is specified shuffling is done at the start of every epoch. With large datasets, this ensures that the order of the training data is different for each epoch, and it helps reduce bias and possible overfitting. In a multi-node training job when `ShuffleConfig` is combined with `S3DataDistributionType` of `ShardedByS3Key`, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

**Contents**

**Seed**

Determines the shuffling order in `ShuffleConfig` value.

*Type: Long*

*Required: Yes*

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SourceAlgorithm

Service: Amazon SageMaker Service

Specifies an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Contents

AlgorithmName

The name of an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\//)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?!-)$

Required: Yes

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*)$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SourceAlgorithmSpecification
Service: Amazon SageMaker Service

A list of algorithms that were used to create a model package.

Contents

SourceAlgorithms

A list of the algorithms that were used to create a model package.

Type: Array of SourceAlgorithm (p. 1511) objects

Array Members: Fixed number of 1 item.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
StoppingCondition
Service: Amazon SageMaker Service

Specifies a limit to how long a model training or compilation job can run. It also specifies how long you are willing to wait for a managed spot training job to complete. When the job reaches the time limit, Amazon SageMaker ends the training or compilation job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

The training algorithms provided by Amazon SageMaker automatically save the intermediate results of a model training job when possible. This attempt to save artifacts is only a best effort case as model might not be in a state from which it can be saved. For example, if training has just started, the model might not be ready to save. When saved, this intermediate data is a valid model artifact. You can use it to create a model with CreateModel.

**Note**
The Neural Topic Model (NTM) currently does not support saving intermediate model artifacts. When training NTMs, make sure that the maximum runtime is sufficient for the training job to complete.

Contents

MaxRuntimeInSeconds

The maximum length of time, in seconds, that the training or compilation job can run. If job does not complete during this time, Amazon SageMaker ends the job. If value is not specified, default value is 1 day. The maximum value is 28 days.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

MaxWaitTimeInSeconds

The maximum length of time, in seconds, how long you are willing to wait for a managed spot training job to complete. It is the amount of time spent waiting for Spot capacity plus the amount of time the training job runs. It must be equal to or greater than MaxRuntimeInSeconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SubscribedWorkteam
Service: Amazon SageMaker Service

Describes a work team of a vendor that does the a labelling job.

Contents

ListingId
Type: String
Required: No

MarketplaceDescription
The description of the vendor from the Amazon Marketplace.
Type: String
Pattern: .+
Required: No

MarketplaceTitle
The title of the service provided by the vendor in the Amazon Marketplace.
Type: String
Pattern: .+
Required: No

SellerName
The name of the vendor in the Amazon Marketplace.
Type: String
Required: No

WorkteamArn
The Amazon Resource Name (ARN) of the vendor that you have subscribed.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**SuggestionQuery**
Service: Amazon SageMaker Service

Specified in the GetSearchSuggestions (p. 1095) request. Limits the property names that are included in the response.

**Contents**

**PropertyNameQuery**

Defines a property name hint. Only property names that begin with the specified hint are included in the response.

Type: PropertyNameQuery (p. 1488) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Tag
Service: Amazon SageMaker Service

Describes a tag.

Contents

Key

The tag key.
Type: String
Pattern: ^([^\p{L}\p{Z}\p{N}_-/:=+\-@]*)$ 
Required: Yes

Value

The tag value.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 256.
Pattern: ^([^\p{L}\p{Z}\p{N}_-/:=+\-@]*)$ 
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TensorBoardAppSettings
Service: Amazon SageMaker Service

The TensorBoard app settings.

Contents

DefaultResourceSpec

The instance type and quantity.

Type: ResourceSpec (p. 1499) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TensorBoardOutputConfig
Service: Amazon SageMaker Service

Configuration of storage locations for TensorBoard output.

Contents

LocalPath

Path to local storage location for tensorBoard output. Defaults to /opt/ml/output/tensorboard.

Type: String

Length Constraints: Maximum length of 4096.

Pattern: .*

Required: No

S3OutputPath

Path to Amazon S3 storage location for TensorBoard output.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/*(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrainingJob**

Service: Amazon SageMaker Service

Contains information about a training job.

**Contents**

**AlgorithmSpecification**

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 1274) object

Required: No

**AutoMLJobArn**

The Amazon Resource Name (ARN) of the job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:automl-job/.*

Required: No

**BillableTimeInSeconds**

The billable time in seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**CheckpointConfig**

Contains information about the output location for managed spot training checkpoint data.

Type: CheckpointConfig (p. 1314) object

Required: No

**CreationTime**

A timestamp that indicates when the training job was created.

Type: Timestamp

Required: No

**DebugHookConfig**

Configuration information for the debug hook parameters, collection configuration, and storage paths.

Type: DebugHookConfig (p. 1331) object

Required: No

**DebugRuleConfigurations**

Information about the debug rule configuration.
DebugRuleConfiguration

Type: Array of DebugRuleConfiguration (p. 1333) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

DebugRuleEvaluationStatuses

Information about the evaluation status of the rules for the training job.

Type: Array of DebugRuleEvaluationStatus (p. 1335) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

EnableInterContainerTrafficEncryption

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableManagedSpotTraining

When true, enables managed spot training using Amazon EC2 Spot instances to run training jobs instead of on-demand instances. For more information, see Managed Spot Training in Amazon SageMaker (p. 599).

Type: Boolean

Required: No

EnableNetworkIsolation

If the TrainingJob was created with network isolation, the value is set to true. If network isolation is enabled, nodes can't communicate beyond the VPC they run in.

Type: Boolean

Required: No

ExperimentConfig

Configuration for the experiment.

Type: ExperimentConfig (p. 1348) object

Required: No

FailureReason

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

FinalMetricDataList

A list of final metric values that are set when the training job completes. Used only if the training job was configured to use metrics.
Type: Array of MetricData (p. 1424) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

**HyperParameters**

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: .*

Value Length Constraints: Maximum length of 256.

Value Pattern: .*

Required: No

**InputDataConfig**

An array of Channel objects that describes each data input channel.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

**LabelingJobArn**

The Amazon Resource Name (ARN) of the labeling job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*

Required: No

**LastModifiedTime**

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

Required: No

**ModelArtifacts**

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 1426) object

Required: No

**OutputDataConfig**

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: OutputDataConfig (p. 1466) object
Required: No

**ResourceConfig**

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: ResourceConfig (p. 1496) object

Required: No

**RoleArn**

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_]/+$`

Required: No

**SecondaryStatus**

Provides detailed information about the state of the training job. For detailed information about the secondary status of the training job, see [StatusMessage under SecondaryStatusTransition (p. 1507)](https://docs.aws.amazon.com/sagemaker/latest/dg/secondary-status-message.html).

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

- **InProgress**
  - Starting - Starting the training job.
  - Downloading - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
  - Training - Training is in progress.
  - Uploading - Training is complete and the model artifacts are being uploaded to the S3 location.

- **Completed**
  - Completed - The training job has completed.

- **Failed**
  - Failed - The training job has failed. The reason for the failure is returned in the FailureReason field of `DescribeTrainingJobResponse`.

- **Stopped**
  - MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed runtime.
  - Stopped - The training job has stopped.

- **Stopping**
  - Stopping - Stopping the training job.

**Important**

Valid values for `SecondaryStatus` are subject to change.

We no longer support the following secondary statuses:

- LaunchingMLInstances
- PreparingTrainingStack
- DownloadingTrainingImage

Type: String
Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed | Interrupted | MaxWaitTimeExceeded

Required: No

**SecondaryStatusTransitions**

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 1507) objects

Required: No

**StoppingCondition**

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts, so the results of training are not lost.

Type: StoppingCondition (p. 1513) object

Required: No

**Tags**

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 1517) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**TensorBoardOutputConfig**

Configuration of storage locations for TensorBoard output.

Type: TensorBoardOutputConfig (p. 1519) object

Required: No

**TrainingEndTime**

Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of TrainingStartTime and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

**TrainingJobArn**

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Required: No

**TrainingJobName**

The name of the training job.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

**TrainingJobStatus**

The status of the training job.

Training job statuses are:

- **InProgress** - The training is in progress.
- **Completed** - The training job has completed.
- **Failed** - The training job has failed. To see the reason for the failure, see the FailureReason field in the response to a DescribeTrainingJobResponse call.
- **Stopping** - The training job is stopping.
- **Stopped** - The training job has stopped.

For more detailed information, see SecondaryStatus.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

**TrainingStartTime**

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of TrainingEndTime. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

Required: No

**TrainingTimeInSeconds**

The training time in seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**TuningJobArn**

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String
Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

Required: No

VpcConfig

A VpcConfig (p. 1577) object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 1577) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobDefinition
Service: Amazon SageMaker Service

Defines the input needed to run a training job using the algorithm.

Contents

HyperParameters

The hyperparameters used for the training job.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Key Pattern: .*

Value Length Constraints: Maximum length of 256.

Value Pattern: .*

Required: No

InputDataConfig

An array of Channel objects, each of which specifies an input source.

Type: Array of Channel (p. 1310) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: Yes

OutputDataConfig

the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 1466) object

Required: Yes

ResourceConfig

The resources, including the ML compute instances and ML storage volumes, to use for model training.

Type: ResourceConfig (p. 1496) object

Required: Yes

StoppingCondition

Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms can use this 120-second window to save the model artifacts.

Type: StoppingCondition (p. 1513) object

Required: Yes
TrainingInputMode

The input mode used by the algorithm for the training job. For the input modes that Amazon SageMaker algorithms support, see Algorithms.

If an algorithm supports the File input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the Pipe input mode, Amazon SageMaker streams data directly from S3 to the container.

Type: String

Valid Values: Pipe | File

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobStatusCounters
Service: Amazon SageMaker Service

The numbers of training jobs launched by a hyperparameter tuning job, categorized by status.

Contents

Completed
The number of completed training jobs launched by the hyperparameter tuning job.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

InProgress
The number of in-progress training jobs launched by a hyperparameter tuning job.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

NonRetryableError
The number of training jobs that failed and can't be retried. A failed training job can't be retried if it failed because a client error occurred.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

RetryableError
The number of training jobs that failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

Stopped
The number of training jobs launched by a hyperparameter tuning job that were manually stopped.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobSummary
Service: Amazon SageMaker Service

Provides summary information about a training job.

Contents

CreationTime

A timestamp that shows when the training job was created.

Type: Timestamp

Required: Yes

LastModifiedTime

Timestamp when the training job was last modified.

Type: Timestamp

Required: No

TrainingEndTime

A timestamp that shows when the training job ended. This field is set only if the training job has one of the terminal statuses (Completed, Failed, or Stopped).

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Required: Yes

TrainingJobName

The name of the training job that you want a summary for.

Type: String


Pattern: ^[a-zA-Z0-9\-]+\*[a-zA-Z0-9\-]*$*

Required: Yes

TrainingJobStatus

The status of the training job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingSpecification
Service: Amazon SageMaker Service

Defines how the algorithm is used for a training job.

Contents

MetricDefinitions

A list of MetricDefinition objects, which are used for parsing metrics generated by the algorithm.

Type: Array of MetricDefinition (p. 1425) objects

Array Members: Minimum number of 0 items. Maximum number of 40 items.

Required: No

SupportedHyperParameters

A list of the HyperParameterSpecification objects, that define the supported hyperparameters. This is required if the algorithm supports automatic model tuning.

Type: Array of HyperParameterSpecification (p. 1381) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

Required: No

SupportedTrainingInstanceTypes

A list of the instance types that this algorithm can use for training.

Type: Array of strings

Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.p3dn.24xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge

Required: Yes

SupportedTuningJobObjectiveMetrics

A list of the metrics that the algorithm emits that can be used as the objective metric in a hyperparameter tuning job.

Type: Array of HyperParameterTuningJobObjective (p. 1391) objects

Required: No

SupportsDistributedTraining

Indicates whether the algorithm supports distributed training. If set to false, buyers can't request more than one instance during training.

Type: Boolean

Required: No
TrainingChannels

A list of ChannelSpecification objects, which specify the input sources to be used by the algorithm.

Type: Array of ChannelSpecification (p. 1312) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: Yes

TrainingImage

The Amazon ECR registry path of the Docker image that contains the training algorithm.

Type: String

Length Constraints: Maximum length of 255.

Pattern: [\S]+

Required: Yes

TrainingImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: ^[Ss][Hh][Aa]256:[0-9a-fA-F]{64}$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformDataSource
Service: Amazon SageMaker Service

Describes the location of the channel data.

Contents

S3DataSource

The S3 location of the data source that is associated with a channel.

Type: TransformS3DataSource (p. 1546) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformInput

Service: Amazon SageMaker Service

Describes the input source of a transform job and the way the transform job consumes it.

Contents

CompressionType

If your transform data is compressed, specify the compression type. Amazon SageMaker automatically decompresses the data for the transform job accordingly. The default value is None.

Type: String

Valid Values: None | Gzip

Required: No

ContentType

The multipurpose internet mail extension (MIME) type of the data. Amazon SageMaker uses the MIME type with each http call to transfer data to the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: *

Required: No

DataSource

Describes the location of the channel data, which is, the S3 location of the input data that the model can consume.

Type: TransformDataSource (p. 1535) object

Required: Yes

SplitType

The method to use to split the transform job's data files into smaller batches. Splitting is necessary when the total size of each object is too large to fit in a single request. You can also use data splitting to improve performance by processing multiple concurrent mini-batches. The default value for SplitType is None, which indicates that input data files are not split, and request payloads contain the entire contents of an input object. Set the value of this parameter to Line to split records on a newline character boundary. SplitType also supports a number of record-oriented binary data formats.

When splitting is enabled, the size of a mini-batch depends on the values of the BatchStrategy and MaxPayloadInMB parameters. When the value of BatchStrategy is MultiRecord, Amazon SageMaker sends the maximum number of records in each request, up to the MaxPayloadInMB limit. If the value of BatchStrategy is SingleRecord, Amazon SageMaker sends individual records in each request.

Note

Some data formats represent a record as a binary payload wrapped with extra padding bytes. When splitting is applied to a binary data format, padding is removed if the value of BatchStrategy is set to SingleRecord. Padding is not removed if the value of BatchStrategy is set to MultiRecord.
For more information about RecordIO, see Create a Dataset Using RecordIO in the MXNet documentation. For more information about TFRecord, see Consuming TFRecord data in the TensorFlow documentation.

Type: String

Valid Values: None | Line | RecordIO | TFRecord

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformJobDefinition
Service: Amazon SageMaker Service

Defines the input needed to run a transform job using the inference specification specified in the algorithm.

Contents

BatchStrategy

A string that determines the number of records included in a single mini-batch.

SingleRecord means only one record is used per mini-batch. MultiRecord means a mini-batch is set to contain as many records that can fit within the MaxPayloadInMB limit.

Type: String

Valid Values: MultiRecord | SingleRecord

Required: No

Environment

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 10240.

Value Pattern: \[\S\s\]*

Required: No

MaxConcurrentTransforms

The maximum number of parallel requests that can be sent to each instance in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MaxPayloadInMB

The maximum payload size allowed, in MB. A payload is the data portion of a record (without metadata).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

TransformInput

A description of the input source and the way the transform job consumes it.
Type: TransformInput (p. 1536) object

Required: Yes

TransformOutput

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: TransformOutput (p. 1542) object

Required: Yes

TransformResources

Identifies the ML compute instances for the transform job.

Type: TransformResources (p. 1544) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TransformJobSummary**  
Service: Amazon SageMaker Service

Provides a summary of a transform job. Multiple `TransformJobSummary` objects are returned as a list after in response to a `ListTransformJobs` (p. 1178) call.

**Contents**

**CreationTime**

A timestamp that shows when the transform Job was created.

Type: Timestamp  
Required: Yes

**FailureReason**

If the transform job failed, the reason it failed.

Type: String  
Length Constraints: Maximum length of 1024.  
Required: No

**LastModifiedTime**

Indicates when the transform job was last modified.

Type: Timestamp  
Required: No

**TransformEndTime**

Indicates when the transform job ends on compute instances. For successful jobs and stopped jobs, this is the exact time recorded after the results are uploaded. For failed jobs, this is when Amazon SageMaker detected that the job failed.

Type: Timestamp  
Required: No

**TransformJobArn**

The Amazon Resource Name (ARN) of the transform job.

Type: String  
Length Constraints: Maximum length of 256.  
**Pattern:** `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`  
Required: Yes

**TransformJobName**

The name of the transform job.

Type: String  
**Pattern:** `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
Required: Yes

**TransformJobStatus**

The status of the transform job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformOutput
Service: Amazon SageMaker Service

Describes the results of a transform job.

Contents

Accept
The MIME type used to specify the output data. Amazon SageMaker uses the MIME type with each http call to transfer data from the transform job.

Type: String
Length Constraints: Maximum length of 256.
Pattern: .*
Required: No

AssembleWith
Defines how to assemble the results of the transform job as a single S3 object. Choose a format that is most convenient to you. To concatenate the results in binary format, specify None. To add a newline character at the end of every transformed record, specify Line.

Type: String
Valid Values: None | Line
Required: No

KmsKeyId
The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The KmsKeyId can be any of the following formats:
- Key ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- Key ARN: arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- Alias name: alias/ExampleAlias

If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. For more information, see KMS-Managed Encryption Keys in the Amazon Simple Storage Service Developer Guide.

The KMS key policy must grant permission to the IAM role that you specify in your CreateModel (p. 902) request. For more information, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Type: String
Length Constraints: Maximum length of 2048.
Pattern: .*
Required: No
S3OutputPath

The Amazon S3 path where you want Amazon SageMaker to store the results of the transform job. For example, `s3://bucket-name/key-name-prefix`.

For every S3 object used as input for the transform job, batch transform stores the transformed data with an `.out` suffix in a corresponding subfolder in the location in the output prefix. For example, for the input data stored at `s3://bucket-name/input-name-prefix/dataset01/data.csv`, batch transform stores the transformed data at `s3://bucket-name/output-name-prefix/input-name-prefix/data.csv.out`. Batch transform doesn't upload partially processed objects. For an input S3 object that contains multiple records, it creates an `.out` file only if the transform job succeeds on the entire file. When the input contains multiple S3 objects, the batch transform job processes the listed S3 objects and uploads only the output for successfully processed objects. If any object fails in the transform job batch transform marks the job as failed to prompt investigation.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?.*$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformResources
Service: Amazon SageMaker Service

Describes the resources, including ML instance types and ML instance count, to use for transform job.

Contents

InstanceCount

The number of ML compute instances to use in the transform job. For distributed transform jobs, specify a value greater than 1. The default value is 1.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType

The ML compute instance type for the transform job. If you are using built-in algorithms to transform moderately sized datasets, we recommend using ml.m4.xlarge or ml.m5.large instance types.

Type: String

Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge
| ml.m4.16xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.8xlarge
| ml.c4.xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge
| ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge
| ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge
| ml.m5.large | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge
| ml.m5.24xlarge

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt model data on the storage volume attached to the ML compute instance(s) that run the batch transform job. The VolumeKmsKeyId can be any of the following formats:

- Key ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- Key ARN: arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- Alias name: alias/ExampleAlias

Type: String

Length Constraints: Maximum length of 2048.

Pattern: .*

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformS3DataSource
Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

S3DataType

If you choose `S3Prefix`, `S3Uri` identifies a key name prefix. Amazon SageMaker uses all objects with the specified key name prefix for batch transform.

If you choose `ManifestFile`, `S3Uri` identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for batch transform.

The following values are compatible: `ManifestFile`, `S3Prefix`

The following value is not compatible: `AugmentedManifestFile`

Type: String

Valid Values: `ManifestFile` | `S3Prefix` | `AugmentedManifestFile`

Required: Yes

S3Uri

Depending on the value specified for the `S3DataType`, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: `s3://bucketname/exampleprefix`.
- A manifest might look like this: `s3://bucketname/example.manifest`

The manifest is an S3 object which is a JSON file with the following format:

```json
[ {
    "prefix": "s3://customer_bucket/some/prefix/",
    "relative/path/to/custdata-1",
    "relative/path/custdata-2",
    ...
    "relative/path/custdata-N"
}
]
```

The preceding JSON matches the following `S3Uris`:

- `s3://customer_bucket/some/prefix/relative/path/to/custdata-1`
- `s3://customer_bucket/some/prefix/relative/path/custdata-2`
- ...
- `s3://customer_bucket/some/prefix/relative/path/custdata-N`

The complete set of `S3Uris` in this manifest constitutes the input data for the channel for this datasource. The object that each `S3Uris` points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String
Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*$/

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Trial
Service: Amazon SageMaker Service

A summary of the properties of a trial as returned by the Search (p. 1196) API.

Contents

CreatedBy
Information about the user who created or modified an experiment, trial, or trial component.
Type: UserContext (p. 1572) object
Required: No

CreationTime
When the trial was created.
Type: Timestamp
Required: No

DisplayName
The name of the trial as displayed. If DisplayName isn't specified, TrialName is displayed.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: No

ExperimentName
The name of the experiment the trial is part of.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: No

LastModifiedBy
Information about the user who created or modified an experiment, trial, or trial component.
Type: UserContext (p. 1572) object
Required: No

LastModifiedTime
Who last modified the trial.
Type: Timestamp
Required: No

Source
The source of the trial.
**Type:** TrialSource (p. 1564) object

**Required:** No

**Tags**

The list of tags that are associated with the trial. You can use Search (p. 1196) API to search on the tags.

**Type:** Array of Tag (p. 1517) objects

**Array Members:** Minimum number of 0 items. Maximum number of 50 items.

**Required:** No

**TrialArn**

The Amazon Resource Name (ARN) of the trial.

**Type:** String

**Length Constraints:** Maximum length of 256.

**Pattern:** arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial/.*

**Required:** No

**TrialComponentSummaries**

A list of the components associated with the trial. For each component, a summary of the component's properties is included.

**Type:** Array of TrialComponentSimpleSummary (p. 1557) objects

**Required:** No

**TrialName**

The name of the trial.

**Type:** String

**Length Constraints:** Minimum length of 1. Maximum length of 82.

**Pattern:** ^[a-zA-Z0-9]-[^a-zA-Z0-9]*[^a-zA-Z0-9\-][a-zA-Z0-9\-]*$

**Required:** No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponent**

Service: Amazon SageMaker Service

A summary of the properties of a trial component as returned by the Search (p. 1196) API.

**Contents**

**CreatedBy**

Information about the user who created or modified an experiment, trial, or trial component.

Type: `UserContext (p. 1572)` object

Required: No

**CreationTime**

When the component was created.

Type: `Timestamp`

Required: No

**DisplayName**

The name of the component as displayed. If `DisplayName` isn't specified, `TrialComponentName` is displayed.

Type: `String`

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**EndTime**

When the component ended.

Type: `Timestamp`

Required: No

**InputArtifacts**

The input artifacts of the component.

Type: `String to TrialComponentArtifact (p. 1553)` object map

Key Length Constraints: Maximum length of 64.

Key Pattern: `. *`

Required: No

**LastModifiedBy**

Information about the user who created or modified an experiment, trial, or trial component.

Type: `UserContext (p. 1572)` object

Required: No
LastModifiedTime

When the component was last modified.
Type: Timestamp
Required: No

Metrics

The metrics for the component.
Type: Array of TrialComponentMetricSummary (p. 1554) objects
Required: No

OutputArtifacts

The output artifacts of the component.
Type: String to TrialComponentArtifact (p. 1553) object map
Key Length Constraints: Maximum length of 64.
Key Pattern: .*
Required: No

Parameters

The hyperparameters of the component.
Type: String to TrialComponentParameterValue (p. 1556) object map
Key Length Constraints: Maximum length of 256.
Key Pattern: .*
Required: No

Parents

An array of the parents of the component. A parent is a trial the component is associated with and the experiment the trial is part of. A component might not have any parents.
Type: Array of Parent (p. 1470) objects
Required: No

Source

The source of the trial component.
Type: TrialComponentSource (p. 1559) object
Required: No

SourceDetail

The source of the trial component.
Type: TrialComponentSourceDetail (p. 1560) object
Required: No

StartTime

When the component started.
Type: Timestamp
Required: No

**Status**

The status of the trial component.

Type: `TrialComponentStatus (p. 1561)` object
Required: No

**Tags**

The list of tags that are associated with the component. You can use `Search (p. 1196)` API to search on the tags.

Type: Array of `Tag (p. 1517)` objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**TrialComponentArn**

The Amazon Resource Name (ARN) of the trial component.

Type: String
Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*`
Required: No

**TrialComponentName**

The name of the trial component.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9][\-\*][a-zA-Z0-9\-]*$`
Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentArtifact**

Service: Amazon SageMaker Service

Represents an input or output artifact of a trial component. You specify TrialComponentArtifact as part of the InputArtifacts and OutputArtifacts parameters in the CreateTrialComponent (p. 947) request.

Examples of input artifacts are datasets, algorithms, hyperparameters, source code, and instance types. Examples of output artifacts are metrics, snapshots, logs, and images.

**Contents**

**MediaType**

The media type of the artifact, which indicates the type of data in the artifact file. The media type consists of a *type* and a *subtype* concatenated with a slash (/) character, for example, text/csv, image/jpeg, and s3/uri. The type specifies the category of the media. The subtype specifies the kind of data.

Type: String

Length Constraints: Maximum length of 64.

Pattern: `^[\w]+/\[\w+]+$`

Required: No

**Value**

The location of the artifact.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `.*`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentMetricSummary**

Service: Amazon SageMaker Service

A summary of the metrics of a trial component.

**Contents**

**Avg**
- The average value of the metric.
- Type: Double
- Required: No

**Count**
- The number of samples used to generate the metric.
- Type: Integer
- Required: No

**Last**
- The most recent value of the metric.
- Type: Double
- Required: No

**Max**
- The maximum value of the metric.
- Type: Double
- Required: No

**MetricName**
- The name of the metric.
- Type: String
- Pattern: .+
- Required: No

**Min**
- The minimum value of the metric.
- Type: Double
- Required: No

**SourceArn**
- The Amazon Resource Name (ARN) of the source.
- Type: String
- Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-\:]*:\[0-9\]{12}::*
Required: No

**StdDev**
The standard deviation of the metric.
Type: Double
Required: No

**TimeStamp**
When the metric was last updated.
Type: Timestamp
Required: No

**See Also**
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentParameterValue**

Service: Amazon SageMaker Service

The value of a hyperparameter. Only one of `NumberValue` or `StringValue` can be specified.

This object is specified in the [CreateTrialComponent](p. 947) request.

**Contents**

**NumberValue**

The numeric value of a numeric hyperparameter. If you specify a value for this parameter, you can't specify the `StringValue` parameter.

Type: Double

Required: No

**StringValue**

The string value of a categorical hyperparameter. If you specify a value for this parameter, you can't specify the `NumberValue` parameter.

Type: String

Length Constraints: Maximum length of 256.

Pattern: . *

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentSimpleSummary**

Service: Amazon SageMaker Service

A short summary of a trial component.

**Contents**

**CreatedBy**

Information about the user who created or modified an experiment, trial, or trial component.

Type: UserContext (p. 1572) object

Required: No

**CreationTime**

When the component was created.

Type: Timestamp

Required: No

**TrialComponentArn**

The Amazon Resource Name (ARN) of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*

Required: No

**TrialComponentName**

The name of the trial component.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: ^[a-zA-Z0-9\-\[\]_\.(\d+)]\*(\-[a-zA-Z0-9\-\[\]_\.(\d+)\])*\*

Required: No

**TrialComponentSource**

The source of the trial component.

Type: TrialComponentSource (p. 1559) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentSource**

Service: Amazon SageMaker Service

The source of the trial component.

**Contents**

**SourceArn**

The Amazon Resource Name (ARN) of the source.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-zA-Z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:.*`

Required: Yes

**SourceType**

The source job type.

Type: String

Length Constraints: Maximum length of 128.

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrialComponentSourceDetail
Service: Amazon SageMaker Service

Detailed information about the source of a trial component.

Contents

SourceArn

The Amazon Resource Name (ARN) of the source.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:.*`

Required: No

TrainingJob

Contains information about a training job.

Type: TrainingJob (p. 1520) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentStatus**

Service: Amazon SageMaker Service

The status of the trial component.

**Contents**

**Message**

If the component failed, a message describing why.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: .*

Required: No

**PrimaryStatus**

The status of the trial component.

Type: String

Valid Values: InProgress | Completed | Failed

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialComponentSummary**
Service: Amazon SageMaker Service

A summary of the properties of a trial component. To get all the properties, call the **DescribeTrialComponent** (p. 1083) API and provide the `TrialComponentName`.

**Contents**

**CreatedBy**

Who created the component.

Type: UserContext (p. 1572) object

Required: No

**CreationTime**

When the component was created.

Type: Timestamp

Required: No

**DisplayName**

The name of the component as displayed. If `DisplayName` isn't specified, `TrialComponentName` is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**EndTime**

When the component ended.

Type: Timestamp

Required: No

**LastModifiedBy**

Who last modified the component.

Type: UserContext (p. 1572) object

Required: No

**LastModifiedTime**

When the component was last modified.

Type: Timestamp

Required: No

**StartTime**

When the component started.
Type: Timestamp
Required: No

**Status**

The status of the component. States include:

- InProgress
- Completed
- Failed

Type: `TrialComponentStatus (p. 1561)` object
Required: No

**TrialComponentArn**

The ARN of the trial component.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:experiment-trial-component/.*`

Required: No

**TrialComponentName**

The name of the trial component.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9\-]*[^a-zA-Z0-9\-]`

Required: No

**TrialComponentSource**

The source of the trial component.

Type: `TrialComponentSource (p. 1559)` object
Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialSource**

Service: Amazon SageMaker Service

The source of the trial.

**Contents**

**SourceArn**

The Amazon Resource Name (ARN) of the source.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-\*:\[0-9]*\]{12}::`

Required: Yes

**SourceType**

The source job type.

Type: String

Length Constraints: Maximum length of 128.

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TrialSummary**

Service: Amazon SageMaker Service

A summary of the properties of a trial. To get the complete set of properties, call the `DescribeTrial` API and provide the `TrialName`.

**Contents**

**CreationTime**

When the trial was created.

Type: Timestamp

Required: No

**DisplayName**

The name of the trial as displayed. If `DisplayName` isn't specified, `TrialName` is displayed.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**LastModifiedTime**

When the trial was last modified.

Type: Timestamp

Required: No

**TrialArn**

The Amazon Resource Name (ARN) of the trial.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:0-9{12}:experiment-trial/.*`

Required: No

**TrialName**

The name of the trial.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 82.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**TrialSource**

The source of the trial.

Type: `TrialSource` object
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TuningJobCompletionCriteria
Service: Amazon SageMaker Service

The job completion criteria.

Contents

TargetObjectiveMetricValue

The objective metric's value.

Type: Float

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UiConfig
Service: Amazon SageMaker Service

Provided configuration information for the worker UI for a labeling job.

Contents

UiTemplateS3Uri

The Amazon S3 bucket location of the UI template. For more information about the contents of a UI template, see Creating Your Custom Labeling Task Template.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^((https|s3)://[^/]+)([^/]+)/*$?

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UiTemplate
Service: Amazon SageMaker Service

The Liquid template for the worker user interface.

Contents

Content

The content of the Liquid template for the worker user interface.

Type: String


Pattern: \S\s*+ 

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UiTemplateInfo
Service: Amazon SageMaker Service

Container for user interface template information.

Contents

ContentSha256

The SHA 256 hash that you used to create the request signature.

Type: String
Required: No

Url

The URL for the user interface template.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
USD
Service: Amazon SageMaker Service

Represents an amount of money in United States dollars/

Contents

Cents

The fractional portion, in cents, of the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 99.

Required: No

Dollars

The whole number of dollars in the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

TenthFractionsOfACent

Fractions of a cent, in tenths.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 9.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UserContext
Service: Amazon SageMaker Service

Information about the user who created or modified an experiment, trial, or trial component.

Contents

DomainId

The domain associated with the user.

Type: String
Required: No

UserProfileArn

The Amazon Resource Name (ARN) of the user's profile.

Type: String
Required: No

UserProfileName

The name of the user's profile.

Type: String
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UserProfileDetails
Service: Amazon SageMaker Service

The user profile details.

Contents

CreationTime
The creation time.
Type: Timestamp
Required: No

DomainId
The domain ID.
Type: String
Length Constraints: Maximum length of 63.
Required: No

LastModifiedTime
The last modified time.
Type: Timestamp
Required: No

Status
The status.
Type: String
Valid Values: Deleting | Failed | InService | Pending
Required: No

UserProfileName
The user profile name.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9][-]*[a-zA-Z0-9]*$
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
• AWS SDK for Ruby V2
UserSettings
Service: Amazon SageMaker Service
A collection of settings.

Contents

ExecutionRole
The execution role for the user.
Type: String
Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@\-_]+$
Required: No

JupyterServerAppSettings
The Jupyter server's app settings.
Type: JupyterServerAppSettings (p. 1403) object
Required: No

KernelGatewayAppSettings
The kernel gateway app settings.
Type: KernelGatewayAppSettings (p. 1404) object
Required: No

SecurityGroups
The security groups.
Type: Array of strings
Array Members: Maximum number of 5 items.
Length Constraints: Maximum length of 32.
Pattern: [-0-9a-zA-Z]+
Required: No

SharingSettings
The sharing settings.
Type: SharingSettings (p. 1509) object
Required: No

TensorBoardAppSettings
The TensorBoard app settings.
Type: TensorBoardAppSettings (p. 1518) object
Required: No
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
VpcConfig
Service: Amazon SageMaker Service

Specifies a VPC that your training jobs and hosted models have access to. Control access to and from your training and model containers by configuring the VPC. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Contents

SecurityGroupIds

The VPC security group IDs, in the form sg-xxxxxxxx. Specify the security groups for the VPC that is specified in the Subnets field.

Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 5 items.
Length Constraints: Maximum length of 32.
Pattern: [-0-9a-zA-Z]+
Required: Yes

Subnets

The ID of the subnets in the VPC to which you want to connect your training job or model.

Note
Amazon EC2 P3 accelerated computing instances are not available in the c/d/e availability zones of region us-east-1. If you want to create endpoints with P3 instances in VPC mode in region us-east-1, create subnets in a/b/f availability zones instead.

Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 16 items.
Length Constraints: Maximum length of 32.
Pattern: [-0-9a-zA-Z]+
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Workteam
Services: Amazon SageMaker Service

Provides details about a labeling work team.

Contents

CreateDate

The date and time that the work team was created (timestamp).

Type: Timestamp

Required: No

Description

A description of the work team.

Type: String


Pattern: .+

Required: Yes

LastUpdatedDate

The date and time that the work team was last updated (timestamp).

Type: Timestamp

Required: No

MemberDefinitions

The Amazon Cognito user groups that make up the work team.

Type: Array of MemberDefinition (p. 1423) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: Yes

NotificationConfiguration

Configures SNS notifications of available or expiring work items for work teams.

Type: NotificationConfiguration (p. 1463) object

Required: No

ProductListingIds

The Amazon Marketplace identifier for a vendor's work team.

Type: Array of strings

Required: No

SubDomain

The URI of the labeling job's user interface. Workers open this URI to start labeling your data objects.

Type: String
Required: No

**WorkteamArn**

The Amazon Resource Name (ARN) that identifies the work team.

**Type:** String

**Length Constraints:** Maximum length of 256.

**Pattern:** `arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

**WorkteamName**

The name of the work team.

**Type:** String

**Length Constraints:** Minimum length of 1. Maximum length of 63.

**Pattern:** `^[a-zA-Z0-9](\-*[a-zA-Z0-9])*$`

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

### Amazon SageMaker Runtime

Currently Amazon SageMaker Runtime does not support any data types.

### Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

**AccessDeniedException**

You do not have sufficient access to perform this action.

HTTP Status Code: 400

**IncompleteSignature**

The request signature does not conform to AWS standards.

HTTP Status Code: 400

**InternalFailure**

The request processing has failed because of an unknown error, exception or failure.
HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400
ServiceUnavailable
The request has failed due to a temporary failure of the server.
HTTP Status Code: 503

ThrottlingException
The request was denied due to request throttling.
HTTP Status Code: 400

ValidationError
The input fails to satisfy the constraints specified by an AWS service.
HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the Amazon Web Services General Reference.

Action
The action to be performed.
Type: string
Required: Yes

Version
The API version that the request is written for, expressed in the format YYYY-MM-DD.
Type: string
Required: Yes

X-Amz-Algorithm
The hash algorithm that you used to create the request signature.
Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.
Type: string
Valid Values: AWS4-HMAC-SHA256
Required: Conditional

X-Amz-Credential
The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: access_key/YYYYMMDD/region/service/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the Amazon Web Services General Reference.
Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

**X-Amz-Date**

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: 20120325T120000Z.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the Amazon Web Services General Reference.

Type: string

Required: Conditional

**X-Amz-Security-Token**

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the IAM User Guide.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

**X-Amz-Signature**

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

**X-Amz-SignedHeaders**

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional
# Document History for Amazon SageMaker

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring notebook instances</td>
<td>You can use shell scripts to configure notebook instances when you create or start them. For more information, see Customize a Notebook Instance.</td>
<td>May 1, 2018</td>
</tr>
<tr>
<td>Disable direct internet access</td>
<td>You can now disable direct internet access for notebook instances. For more information, see Notebook Instances Are Enabled with Internet Access by Default.</td>
<td>March 15, 2018</td>
</tr>
<tr>
<td>Application Auto Scaling support</td>
<td>Amazon SageMaker now supports Application Auto Scaling for production variants. For information, see Automatically Scaling Amazon SageMaker SageMaker Models</td>
<td>February 28, 2018</td>
</tr>
<tr>
<td>TensorFlow 1.5 and MXNet 1.0 support (p. 1583)</td>
<td>Amazon SageMaker Deep Learning containers now support TensorFlow 1.5 and Apache MXNet 1.0.</td>
<td>February 27, 2018</td>
</tr>
<tr>
<td>Algorithm / Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>BlazingText algorithm</td>
<td>Amazon SageMaker now supports the BlazingText algorithm.</td>
<td>January 18, 2018</td>
</tr>
<tr>
<td>KMS encryption support for training and hosting</td>
<td>Amazon SageMaker now supports KMS encryption for hosting instances and training model artifacts at rest. You can specify an AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to a hosting endpoint by using the KmsKeyId request parameter in a call to CreateEndpointConfig. You can specify an AWS KMS key that Amazon SageMaker uses to encrypt training model artifacts at rest by setting the KmsKeyId field of the OutputDataConfig object you use to configure your training job.</td>
<td>January 17, 2018</td>
</tr>
<tr>
<td>CloudTrail support</td>
<td>Amazon SageMaker now supports logging with AWS CloudTrail.</td>
<td>January 11, 2018</td>
</tr>
<tr>
<td>DeepAR Forecasting algorithm</td>
<td>Amazon SageMaker now supports the DeepAR algorithm for time series forecasting.</td>
<td>January 8, 2018</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.