MLOps Workload Orchestrator

Implementation Guide

MLOps Workload Orchestrator: Implementation Guide

Table of Contents

Deploy a robust pipeline that uses managed automation tools and machine learning (ML) services to simplify ML model development and production

November 2020 (last update (p. 44): May 2022)

The ML lifecycle is an iterative and repetitive process that involves changing models over time and learning from new data. As ML applications gain popularity, organizations are building new and better applications for a wide range of use cases including optimized email campaigns, forecasting tools, recommendation engines, self-driving vehicles, virtual personal assistants, and more. While operational and pipelining processes vary greatly across projects and organizations, the processes contain commonalities across use cases.

This solution helps you streamline and enforce architecture best practices by providing an extendable framework for managing ML pipelines for Amazon Web Services (AWS) ML services and third-party services. The solution's template allows you to train models, upload trained models, configure the orchestration of the pipeline, initiate the start of the deployment process, move models through different stages of deployment, and monitor the successes and failures of the operations. The solution also provides a pipeline for building and registering Docker images for custom algorithms that can be used for model deployment on an Amazon SageMaker endpoint.

You can use batch and real-time data inferences to configure the pipeline for your business context. This solution increases your team's agility and efficiency by allowing them to repeat successful processes at scale.

This solution provides the following key features:

- A preconfigured pipeline initiated through an API call or a Git repository
- Model training using Amazon SageMaker built-in algorithms and training job, hyperparameter tuning job, or autopilot job
- A trained model deployed with an inference endpoint
- Monitoring for deployed machine learning models and detection of any deviation in data quality, model quality, model bias, and/or model explainability
- Support for running your own integration tests to ensure that the deployed model meets expectations
- Provisioning multiple environments to support your ML model's life cycle
- Multi-account support for bring-your-own-model (BYOM) and model monitor pipelines
- Building and registering Docker images for custom algorithms that can be used for model deployment on an SageMaker endpoint
- The option to use SageMaker model registry to deploy versioned models
- User notification of the pipeline outcome though SMS or email

The MLOps Workload Orchestrator solution currently offers 12 pipelines:

- One pipeline to train ML models using SageMaker built-in algorithms and SageMaker training job
- One pipeline to train ML models using SageMaker built-in algorithms and SageMaker hyperparameter tuning job
- One pipeline to train ML models using SageMaker built-in algorithms and SageMaker autopilot job
- Two BYOM real-time inference pipelines for ML models trained using both SageMaker built-in algorithms and custom algorithms
- Two BYOM batch transform pipelines for ML models trained using both SageMaker built-in algorithms and custom algorithms
- One custom algorithm image builder pipeline that can be used to build and register Docker images in Amazon Elastic Container Registry (Amazon ECR) for custom algorithms
- Four model monitor pipelines to continuously monitor the quality of deployed machine learning models by the real-time inference pipeline and alerts for any deviations in data quality, model quality, model bias, and/or model explainability

To support multiple use cases and business needs, this solution provides two AWS CloudFormation templates for single account and multi-account deployments.

- **Template option 1 – Single account**: Use the single-account template to deploy all of the solution's pipelines in the same AWS account. This option is suitable for experimentation, development, and/or small-scale production workloads.
- **Template option 2 – Multi-account**: Use the multi-account template to provision multiple environments (for example, development, staging, and production) across different AWS accounts, which improves governance and increases security and control of the ML pipeline's deployment, provides safe experimentation and faster innovation, and keeps production data and workloads secure and available to ensure business continuity.

This implementation guide describes architectural considerations and configuration steps for deploying MLOps Workload Orchestrator in the AWS Cloud. It includes links to an AWS CloudFormation template that launches and configures the AWS services required to deploy this solution using AWS best practices for security and availability.

The solution is intended for IT infrastructure architects, machine learning engineers, data scientists, developers, DevOps, data analysts, and marketing technology professionals who have practical experience architecting in the AWS Cloud.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of May 2022, the cost for running this solution with the default settings in the US East (N. Virginia) Region is approximately **$374.57 per month**.

Prices are subject to change. For full details, refer to the pricing webpage for each AWS service you will be using in this solution.

Example cost table

The following table provides an example cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region.

The majority of the monthly cost is dependent on AWS Lambda and real-time inferences in Amazon SageMaker. This estimate uses an `ml.m5.large` instance. However, instance type and actual performance is highly dependent on factors like model complexity, algorithm, input size, concurrency, and various other factors.

For cost-efficient performance, you must load test for proper instance size selection and use batch transform instead of real-time inference when possible.

AWS service	Dimensions	Cost per month
Amazon API Gateway	333 million requests (pipelines provisioning and real-time inference requests)	$3.50
AWS Lambda	Requests cost: 333 million requests x $0.20 (per one million requests)	$66.60
	Compute cost: 333,000,000 (runs) x 128/1024 (GB) x 0.1 seconds (run duration) x $0.00001667 (per GB-s)	$69.39
Amazon SageMaker (training job)	One instance (`ml.m5.large`) x $0.115/hour x 1 (hours, job duration) x 20 (jobs per month)	$2.30
Amazon SageMaker (hyperparameter tuning job)	One instance (`ml.m5.large`) x $0.115/hour x 1 (hours, job duration) x 10 (number of candidates) x 20 (jobs per month)	$23.00
Amazon SageMaker (autopilot job)	One instance (`ml.m5.large`) x $0.922/hour x 1 (hours, job duration) x 10 (number of candidates) x 10 (jobs per month)	$92.20

AWS service	Dimensions	Cost per month
Amazon SageMaker (hosting: real-time inference)	One instance (`ml.m5.large`) x $0.115/hour x 24 (hours) x 31 (days)	$85.56
Amazon SageMaker (baseline jobs)	One instance (`ml.m5.large`) x $0.115/hour x 10/60 (hours, job duration) x 2 (jobs per month)	$0.04
Amazon SageMaker (model monitor)	One instance (`ml.m5.large`) x $0.115/hour x 2 (jobs per day) x 10/60 (hours, job duration) x 31 (days)	$1.19
Amazon SageMaker (batch transform)	One instance (`ml.m5.large`) x $0.115 (per hour) x 2 (hours, job duration) x 30 (days)	$6.90
Amazon S3	S3 Standard storage: 100GB x $0.023 (per GB)	$2.30
	PUT requests: 10,000 requests x $0.000005 (per request)	$0.05
	GET requests: 10,000 requests x $0.0000004 (per request)	$0.004
AWS CodePipeline	20 active pipelines x $1.00 (per month)	$20.00
Amazon ECR	Storage: 10 GB x $0.10 (per GB)	$1.00
	Data transfer: 10 GB x $0.02 (per GB)	$0.20
AWS CodeBuild	10 builds per month x 10 minutes (build duration) x $0.0034 (per minute)	$0.34
Total		**$374.57 per month**

Architecture overview

This solution is built with two primary components: 1) the orchestrator component, created by deploying the solution's AWS CloudFormation template, and 2) the AWS CodePipeline instance deployed from either calling the solution's Amazon API Gateway, or by committing a configuration file into an AWS CodeCommit repository. The solution's pipelines are implemented as AWS CloudFormation templates, which allows you to extend the solution and add custom pipelines.

To support multiple use cases and business needs, the solution provides two AWS CloudFormation templates: **option 1** for single-account deployment, and **option 2** for multi-account deployment. In both templates, the solution provides the option to use SageMaker model registry to deploy versioned models. The model registry allows you to catalog models for production, manage model versions, associate metadata with models, manage the approval status of a model, deploy models to production, and automate model deployment with continuous integration and continuous delivery (CI/CD).

Template option 1: Single-account deployment

The solution's single-account architecture provides the following components and workflows, which are shown as numbered steps in Figure 1.

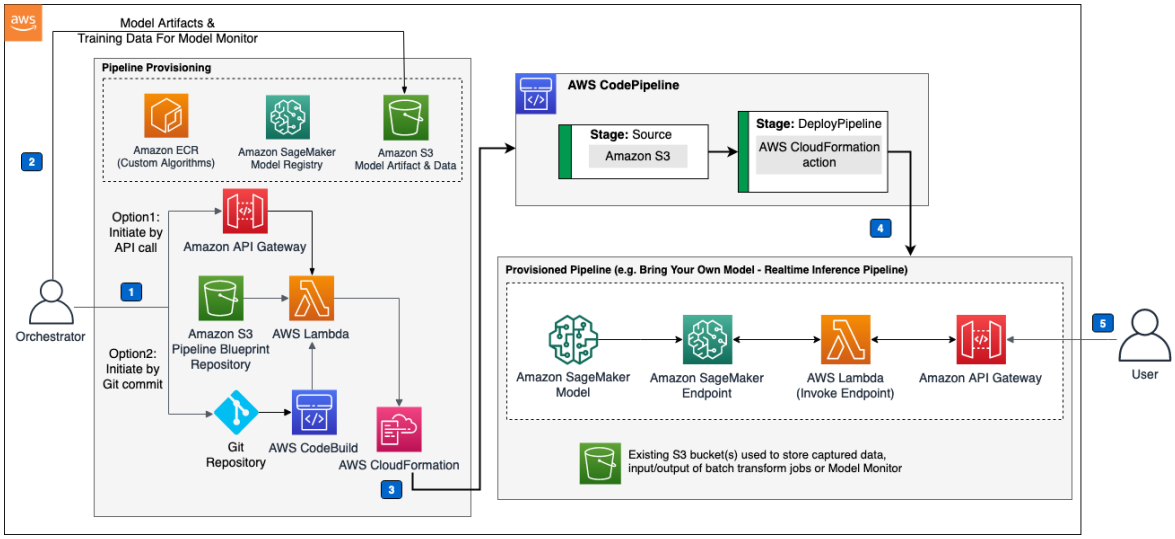

Figure 1: MLOps Workload Orchestrator solution architecture (single account)

This solution's single-account template (Figure 1) provides the following components and workflows:

1. The Orchestrator (solution owner or DevOps engineer) launches the solution in the AWS account and selects the desired options (for example, using SageMaker registry, or providing an existing S3 bucket).
2. The Orchestrator uploads the required assets for the target pipeline (for example, model artifact, training data, and/or custom algorithm zip file) into the S3 assets bucket. If SageMaker model registry is used, the Orchestrator (or an automated pipeline) must register the model with the model registry.
3. A single account AWS CodePipeline instance is provisioned by either sending an API call to the API Gateway, or by committing the `mlops-config.json` file to the Git repository. Depending on the pipeline type, the Orchestrator AWS Lambda function packages the target AWS CloudFormation template and its parameters and configurations using the body of the API call or the `mlops-config.json` file, and uses it as the source stage for the AWS CodePipeline instance.

Note
If you are provisioning the model monitor pipeline, the Orchestrator must first provision the real-time inference pipeline, and then provision the model monitor pipeline.
If a custom algorithm (for example, not a built-in Amazon SageMaker algorithm) was used to train the model, the Orchestrator must provide the Amazon ECR custom algorithm's image URI, or build and register the Docker image using the custom algorithm image builder pipeline.

4. The DeployPipeline stage takes the packaged CloudFormation template and its parameters/configurations and deploys the target pipeline into the same account.
5. After the target pipeline is provisioned, users can access its functionalities. An Amazon Simple Notification Service (Amazon SNS) notification is sent to the email provided in the solution's launch parameters.

Note
The single-account AWS CodePipeline's AWS CloudFormation action is granted admin permissions to deploy different resources by different MLOps pipelines. Roles are defined by the pipelines' CloudFormation templates, allowing the ability to add new pipelines. To restrict the types of resources a template can deploy, customers can create an AWS Identity and Access Management (IAM) role, with limited permissions, and pass it to the CloudFormation action as the deployment role.

Template option 2: Multi-account deployment

This solution uses AWS Organizations and AWS CloudFormation StackSets to allow you to provision or update ML pipelines across AWS accounts and Regions. Using an AWS Organizations delegated administrator account (also referred to as the *Orchestrator account* in this guide) allows you to deploy ML pipelines implemented as AWS CloudFormation templates into selected target accounts (for example, development, staging, and production accounts).

We recommend using AWS Organizations to govern across account deployments with the following structure:

- Orchestrator account (the AWS Organizations delegated administrator account). The MLOps Workload Orchestrator solution is deployed into this account.
- Development Organizational Unit: contains development account(s).
- Staging Organizational Unit: contains staging account(s).
- Production Organizational Unit: contains production account(s).

This solution uses the AWS Organizations service-managed permissions model to allow the Orchestrator account to deploy pipelines into the target accounts (for example, development, staging, and production account).

Note
You must set up the recommended AWS Organizations structure, enable trusted access with AWS Organizations, and register a delegated administrator account before implementing the solution's multi-account deployment option into the Orchestrator account.

Important
By default, the solution expects the Orchestrator account to be an AWS Organizations delegated administrator account. This follows best practices to limit the access to the AWS Organizations management account. However, if you want to use your management account as the Orchestrator account, the solution allows you to switch to the management account by modifying the AWS CloudFormation template parameter: **Are you using a delegated administrator account (AWS Organizations)?** to `No`.

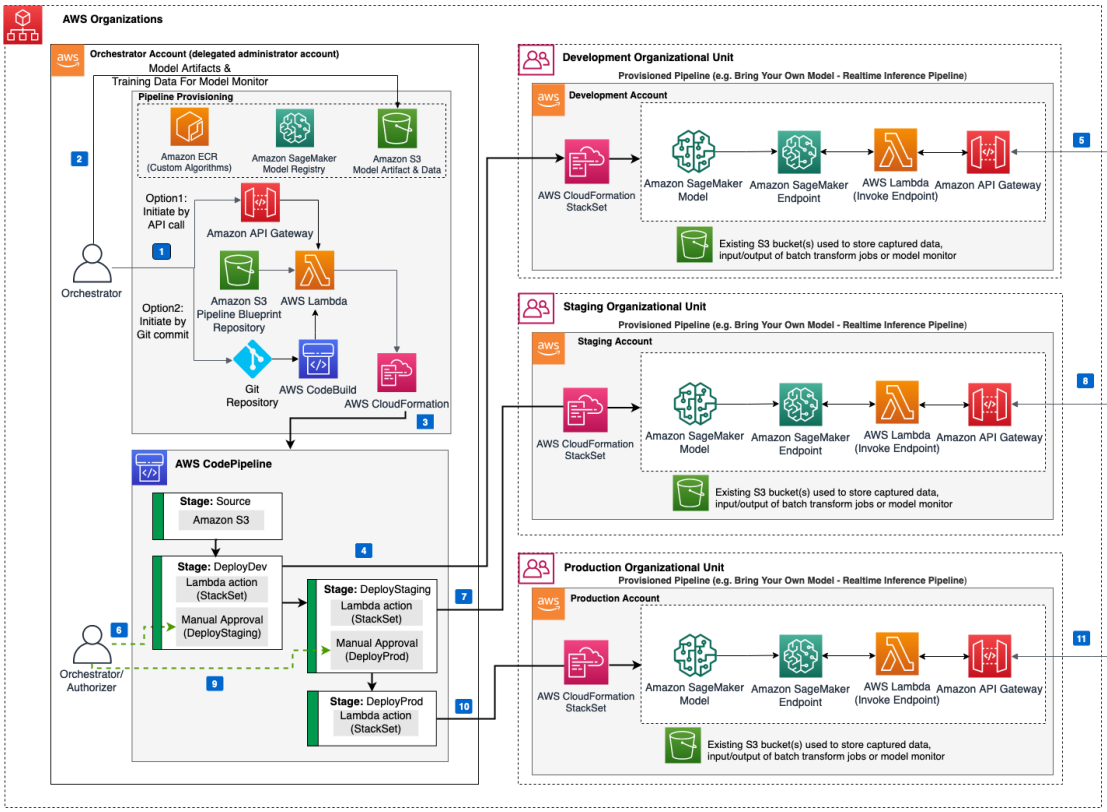

Figure 2: MLOps Workload Orchestrator solution architecture (multi-account)

This solution's multi-account template (Figure 2) provides the following components and workflows:

1. The Orchestrator (solution owner or DevOps engineer with admin access to the orchestrator account) provides the AWS Organizations information (for example, development, staging, and production organizational unit IDs and account numbers). They also specify the desired options (for example, using Amazon SageMaker Registry, or providing an existing S3 bucket), and then launch the solution in their AWS account.
2. The Orchestrator uploads the required assets for the target pipeline (for example, model artifact, training data, and/or custom algorithm zip file) into the S3 assets bucket in the AWS Orchestrator account. If SageMaker model registry is used, the Orchestrator (or an automated pipeline) must register the model with the model registry.
3. A multi-account AWS CodePipeline instance is provisioned by either sending an API call to the API Gateway, or by committing the `mlops-config.json` file to the Git repository. Depending on the pipeline type, the Orchestrator AWS Lambda function packages the target AWS CloudFormation template and its parameters/configurations for each stage using the body of the API call or the `mlops-config.json` file, and uses it as the source stage for the AWS CodePipeline instance.
4. The DeployDev stage takes the packaged CloudFormation template and its parameters/configurations and deploys the target pipeline into the development account.
5. After the target pipeline is provisioned into the development account, the developer can then iterate on the pipeline.
6. After the development is finished, the Orchestrator (or another authorized account) manually approves the DeployStaging action to move to the DeployStaging Stage.
7. The DeployStaging stage deploys the target pipeline into the staging account, using the staging configuration.
8. Testers perform different tests on the deployed pipeline.

9. After the pipeline passes quality tests, the Orchestrator can approve the DeployProd action.
10. The DeployProd stage deploys the target pipeline (with production configurations) into the production account.
11. Finally, the target pipeline is live in production. An Amazon SNS notification is sent to the email provided in the solution's launch parameters.

Note
This solution uses the model's name (provided in the API call or `mlops-config.json` file) as part of the provisioned AWS CloudFormation stack name, which creates the multi-account CodePipeline instance. When a request is made to provision a pipeline, the Orchestrator Lambda first checks to determine whether a stack exists with the specified name. If the stack does not exist, the Lambda function provisions a new stack. If a stack with the same name already exists, the function assumes that you want to update the existing pipeline using the new parameters.

Shared resources and data between accounts

In the multi-account template option, the development, staging, and production accounts each have access to the S3 assets bucket, blueprint bucket, and Amazon ECR repository, and SageMaker model registry in the Orchestrator account. If the S3 assets bucket, Amazon ECR repository, and SageMaker model registry are created by the solution (for example, if the customer did not provide existing resources when installing the solution), the solution will grant permissions to the development, staging, and production accounts access to these resources. If you provided an existing S3 assets bucket and Amazon ECR repository, or are using an SageMaker model registry that was not created by the solution, then you must set up permissions to allow other accounts to access these resources.

The following data is shared across accounts:

- Model artifact
- Baseline datasets used to create baselines for SageMaker data quality, model quality, model bias, and model explainability monitors
- Custom algorithm Amazon ECR image's URL, used to train the model

To allow data separation and security, the following data is not shared between accounts:

- **Location of captured data:** You must provide the full S3 path for each account to store data captured by the real-time inference Amazon SageMaker endpoint.
- **Batch inference data**: You must provide the full S3 path to the inference data for each account.
- **Location of the batch transform's output**: You must provide the full Amazon S3 path for each account where the output of the batch transform job will be stored.
- **Location of baseline job's output**: You must provide the full Amazon S3 path for each account where the output of the baseline job for model monitor will be stored.
- **Location of monitoring schedule job's output**: You must provide the full Amazon S3 path for each account where the output of the monitoring schedule will be stored.

Pipeline descriptions

Model training pipelines

This solution provides three pipelines to train ML models using SageMaker built-in algorithms. Deploying a training pipeline creates the following AWS resources:

- An AWS Lambda function to initiate the creation of SageMaker training, tuning, or autopilot jobs
- An AWS Lambda function to automatically initiate the training Lambda function once the pipeline's CloudFormation template is deployed
- Amazon EventBridge rules to monitor the status of the training jobs
- An Amazon SNS topic, to notify the solution's administrators about pipelines changes via email
- All required AWS Identity and Access Management (IAM) roles

BYOM real-time inference pipelines

This solution allows you to deploy machine learning models trained using Amazon SageMaker built-in algorithms, or custom algorithms on Amazon SageMaker endpoints that provide real-time inferences. Deploying a real-time inference pipeline creates the following AWS resources:

- An Amazon SageMaker model, endpoint configuration, and endpoint
- An AWS Lambda function that invokes the Amazon SageMaker endpoint and returns inferences on the passed data
- An Amazon API Gateway connected to the Lambda that provides authentication and authorization to securely access the Amazon SageMaker endpoint
- All required AWS Identity and Access Management (IAM) roles

BYOM batch transform pipelines

The batch transform pipelines create transform jobs using machine learning models trained using Amazon SageMaker built-in algorithms (or custom algorithms) to perform batch inferences on a batch of data. Deploying a batch transform pipeline creates the following AWS resources:

- An Amazon SageMaker model
- An AWS Lambda function that initiates the creation of the Amazon SageMaker transform job
- All required AWS Identity and Access Management (IAM) roles

Custom algorithm image builder pipeline

The custom algorithm image builder pipeline allows you to use custom algorithms, and build and register Docker images in Amazon ECR. This pipeline is deployed in the Orchestrator account, where the Amazon ECR repository is located. Deploying this pipeline creates the following AWS resources:

- An AWS CodePipeline with the source stage and build stage
  - The build stage uses AWS CodePipeline to build and register the custom images
- All required AWS Identity and Access Management (IAM) roles

Model monitor pipeline

This solution uses Amazon SageMaker model monitor to continuously monitor the quality of deployed machine learning models. The solution supports Amazon SageMaker data quality and model quality, model bias, and model explainability (feature attribution) monitoring. The data from Model Monitor reports can be used to set alerts for violations generated by these monitors. This solution uses the following process to activate continuous model monitoring:

1. The deployed Amazon SageMaker endpoint captures data from incoming requests to the deployed model and the resulting model predictions. The data captured for each deployed model is stored in the S3 bucket location specified by `data_capture_location` in the API call under the prefix *<endpoint-name>*/*<model-variant-name>*/*<year>*/*<month>*/*<day>*/*<hour>*/.

2. For data quality, model bias, and model explainability monitoring, the solution creates baselines from the dataset that was used to train the deployed model. from the dataset that was used to train the deployed model. For model quality monitoring, the baseline dataset contains the predictions of the model and ground truth labels. The baseline datasets must be uploaded to the solution's assets S3 bucket. The datasets S3 keys and the baseline output S3 path must be provided in the API call, or `mlops-config.json` file.
3. For data quality and model quality, the baseline jobs computes metrics and suggests constraints for the metrics and produces two files: `constraints.json` and `statistics.json`. For model bias and model explainability, `analysis_config.json` and `analysis.json` files are generated.
4. The generated json files by baseline jobs are stored in the S3 bucket specified by `baseline_job_output_location` under the prefix `<baseline-job-name>/`. These files are passed as input to the Amazon SageMaker monitors.
5. The solution creates a monitoring schedule job based on your configurations via the API call or `mlops-config.json` file. The monitoring job compares real-time predictions data (captured in the first step) with the baseline created in the previous step (step 2). The job reports for each deployed model monitor pipeline are stored in the S3 bucket location specified by `monitoring_output_location` under the prefix *`<endpoint-name>`*/*`<monitoring-job-name>`*/*`<year>`*/*`<month>`*/*`<day>`*/*`<hour>`*/.

Note
For more information, refer to Amazon SageMaker data quality, model quality, model bias, and model explainability (feature attribution) monitoring.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit the AWS Cloud Security.

IAM roles

AWS Identity and Access Management (IAM) roles allow you to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grant the solution's AWS Lambda functions access to create Regional resources.

AWS Key Management Service (KMS) keys

The MLOps Workload Orchestrator solution allows you to provide your own AWS KMS keys to encrypt captured data by the inference endpoint, model monitor baselines and violations reports, and instances' volumes used by different pipelines. We recommend referring to Security best practices for AWS Key Management Service to enhance the protection of your encryption keys.

Design considerations

Data retention

Depending on the business's requirements and data classification policies, it is advisable to setup S3 Lifecycle Configuration to manage the retention of data captured by the Amazon SageMaker real-time endpoints, such as moving the data to Amazon S3 Glacier for long term archiving.

Bring Your Own Model pipeline

MLOps Workload Orchestrator provisions a pipeline based on the inputs received from either an API call or a Git repository. The provisioned pipeline supports building, deploying, and sharing a machine learning model. However, it does not support training the model. You can customize this solution and *bring your own* training model pipeline.

Custom blueprints/pipelines

This solution supports adding custom blueprints implemented as AWS CloudFormation templates. You can add a custom blueprint with two steps:

1. Create your custom AWS CloudFormation template or AWS Lambda functions
2. Update the Orchestrator Lambda function to add your custom blueprint's logic

Create your custom AWS CloudFormation template or AWS Lambda functions

The MLOps Workload Orchestrator uses AWS Cloud Development Kit (CDK) to generate all its AWS CloudFormation templates and supporting resources such as Lambda functions.

Use the following steps to integrate a custom template:

1. Add your template's CDK code under `source/lib/blueprints/byom/pipeline_definitions`.
2. Make sure your template creates any required roles or permissions. You can reuse existing IAM polices defined in `source/lib/blueprints/byom/pipeline_definitions/iam_policies.py`, or create your own.
3. If your template has Lambda functions, add them under `source/lib/blueprints/byom/lambdas`. Lambda functions are uploaded to the **Blueprints** S3 bucket, under the prefix `blueprints/byom/lambdas`. The name of the Lambda function's folder must be used in your template's CDK code. For example, if your Lambda function folder's name is `myfunction`, you must refer to it as `blueprints/byom/lambdas/myfunction.zip` in your CDK code. For an example, refer to the solution's GitHub.
4. Add your template stack to the CDK's application at `source/app.py`.
5. Add your code to generate the template in `deployment/build-s3-dist.sh`. Follow the documented steps in the build script. Your custom template and any Lambda functions, are packaged with the solution's templates or Lambda functions in the `blueprints.zip` file. The zip file is uploaded to the **Blueprints** S3 bucket when the solution is deployed.

Update the Orchestrator Lambda function to add your custom blueprint's logic

The orchestrator Lambda function is provisions all the solution's blueprints. You must update the Orchestrator Lambda function code at `source/lambdas/pipeline_orchestration/lambda_helpers.py` to add your custom blueprint's template logic.

Use the following steps to update the Orchestrator Lambda function:

1. Update the `template_url` function to add your custom template URL. You must create your own `pipeline_type`, which is used to identify the custom template. For example, `byom_realtime_builtin` is used for the real-time inference with built-in SageMaker algorithms pipeline.
2. Update the `get_stack_name` function to add a stack name for your template.
3. Update the `get_template_parameters` function to get your template's parameters. You can follow the same approach to create a function to get your template's parameters from the API call payload, similar to the functions `get_realtime_specific_params`, `get_batch_specific_params`, `get_model_monitor_params`, etc.
4. Add required APIs call keys to `get_required_keys`. This function is used by the "validate" function to validate that the API call payload contains the required keys (i.e., parameters) used by your template.
5. Add any additional IAM permissions to the `create_orchestrator_policy` function in `source/lib/blueprints/byom/pipeline_definitions/iam_policies.py`.

Regional deployments

This solution uses the AWS CodePipeline and SageMaker services, which are not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS CodePipeline and SageMaker are available. For the most current availability of AWS services by Region, refer to the AWS Regional Services List.

AWS CloudFormation templates

This solution uses AWS CloudFormation to automate deployment. It includes the following two templates—a single-account deployment option, and a multi-account deployment option.

View template

mlops-workload-orchestrator-single-account.template - Use this template to launch the solution with the single-account deployment option. The default configuration deploys two Amazon S3 buckets, an AWS Lambda function, an Amazon API Gateway API, an AWS CodeBuild project, and an Amazon ECR repository. You can customize the template to meet your specific needs.

View template

mlops-workload-orchestrator-multi-account.template - Use this template to launch the solution with the multi-account deployment option. The default configuration deploys two Amazon S3 buckets, an AWS Lambda function, an Amazon API Gateway API, an AWS CodeBuild project, and an Amazon ECR repository. You can customize the template to meet your specific needs.

Automated deployment

Before you launch the solution, review the architecture, configuration, network security, and other considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 3 minutes

Prerequisites

Before you can deploy this solution, ensure that you have access to the following resources:

- A pre-built machine learning model artifact
- A Dockerfile for building a container image for the model artifact if using a custom algorithm. This is not required if you are using prebuilt SageMaker Docker images.
- A tool to call HTTP APIs operations (for example, cURL or Postman), or a tool to commit files to a Git repository.

Deployment overview

Use the following steps to deploy this solution on AWS. For detailed instructions, follow the links for each step.

Template option 1: Single-account deployment

the section called “Step 1. Launch the stack” (p. 15)

the section called “Step 2. Provision the pipeline and deploy the ML model” (p. 18)

the section called “Step 3. Provision the model monitor pipeline (optional)” (p. 19)

Step 1. Launch the stack

> **Important**
> This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy.
> To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your template and deploy the solution. For more information, refer to the Collection of operational metrics (p. 42) section of this guide.

This automated AWS CloudFormation template deploys MLOps Workload Orchestrator in the AWS Cloud. You must have your model artifact’s file and a Dockerfile before launching the stack.

Note
You are responsible for the cost of the AWS services used while running this solution. For more details, visit to the Cost section in this guide, and refer to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and use the button below to launch the `mlops-workload-orchestrator-single-account-framework.template` AWS CloudFormation template.

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

Note
This solution uses the AWS CodePipeline service, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS CodePipeline is available. For the most current availability by Region, refer to the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, refer to IAM and STS Limits in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Email address	*<Requires input>*	Specify an email to receive Amazon SNS notifications about pipeline outcomes.
CodeCommit repository address	<*Optional input*>	Only provide value if you want to provision a pipeline from a CodeCommit repository. For example: `https://git-codecommit.us-east-1.amazonaws.com/v1/repos/`*`<repository-name>`*
Existing S3 bucket name	<*Optional input*>	Optionally, provide the name of an existing S3 bucket to be used as the S3 assets bucket. If an existing bucket is not provided, the solution creates a new S3 bucket. **Note** If you use an existing S3 bucket for the

Parameter	Default	Description
		bucket must meet the following requirements: 1) the bucket must be in the same Region as the MLOps Workload Orchestrator stack, 2) the bucket must allow reading/writing objects to/from the bucket, and 3) versioning must be allowed on the bucket. We recommend blocking public access, enabling S3 server-side encryption, access logging, and secure transport (for example, HTTPS only bucket policy) on your existing S3 bucket.
Existing Amazon ECR repository's name	<*Optional input*>	Optionally, provide the name of an existing Amazon ECR repository name to be used for custom algorithms images. If you do not specify an existing repository, the solution creates a new Amazon ECR repository. **Note** The Amazon ECR repository must be in the same Region where the solution is deployed.
Do you want to use Amazon SageMaker Model Registry?	`No`	By default, this value is `No`. You must provide the algorithm and model artifact location. If you want to use Amazon SageMaker model registry, you must set the value to `Yes`, and provide the model version ARN in the API call. For more details, refer to API operations (p. 27). The solution expects that the model artifact is stored in the S3 assets bucket.

Parameter	Default	Description
Do you want the solution to create an Amazon SageMaker model package group?	`No`	By default, this value is `No`. If you are using Amazon SageMaker model registry, you can set this value to `Yes` to instruct the solution to create a model registry (for example, model package group). Otherwise, you can use your own model registry created outside the solution.
Do you want to allow detailed error message in the APIs response?	`Yes`	By default, this value is `Yes` . If allowed, the API's response returns a detailed message for any server-side error/exception. If you set this parameter to `No` , the API's response returns a general error message.

For more information about creating Amazon SageMaker model registry, setting permissions, and registering models, refer to Register and deploy models with model registry in the *Amazon SageMaker Developer Guide*.

Note
To connect a GitHub or BitBucket code repository to this solution, launch the solution and use the process in the source stage of the pipeline to create GitHubSourceAction and BitBucketSourceAction.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation Console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately three minutes.

   Note
   In addition to the primary AWS Lambda function (`AWSMLOpsFrameworkPipelineOrchestration`), this solution includes the solution-helper Lambda function, which runs only during initial configuration or when resources are updated or deleted.

When you run this solution, you will notice both Lambda functions in the AWS Management Console. Only the `AWSMLOpsFrameworkPipelineOrchestration` function is regularly active. However, you must not delete the `solution-helper` function, as it is necessary to manage associated resources.

Step 2. Provision the pipeline and train or deploy the ML model

Use the following procedure to provision the pipeline and train/deploy your ML model. If you are using API provisioning, the body of the API call must have the information specified in API

operations (p. 27). API endpoints require authentication with IAM. For more information, refer to the How do I enable IAM authentication for API Gateway APIs? Support topic, and the Signing AWS requests with Signature Version 4 topic in the *AWS General Reference Guide*.

Note
If you are using API provisioning to launch the stack, you must make a POST request to the API Gateway endpoint specified in the stack's output. The path will be structured as *<apigateway_endpoint>*/provisionpipeline.
If you are using Git provisioning to launch the stack, you must create a file named `mlops-config.json` and commit the file to the repository's main branch.

1. Monitor the progress of the pipeline by calling the *<apigateway_endpoint>*`/pipelinestatus`. The pipeline_id is displayed in the response of the initial /provisionpipeline API call.
2. Run the provisioned pipeline by uploading the model artifacts to the S3 bucket specified in the output of the CloudFormation stack of the pipeline.

When the pipeline provisioning is complete, you will receive another apigateway_endpoint as the inference endpoint of the deployed model.

Step 3. Provision the model monitor pipeline (optional)

Use the following procedure to provision the pipeline and deploy Amazon SageMaker data quality, model quality, model bias, and/or model explainability monitors. If you use API provisioning, the body of the API call must have the information specified in API operations (p. 27).

Note
If you use API provisioning to launch the stack, you must make a POST request to the API Gateway endpoint specified in the stack's output. The path will be structured as *<apigateway_endpoint>*`/provisionpipeline`.
If you are using Git provisioning to launch the stack, you must create a file named `mlops-config.json` and commit the file to the repository's main branch.

1. Monitor the progress of the pipeline by calling the *<apigateway_endpoint>*`/pipelinestatus`. The `pipeline_id` is displayed in the response of the initial `/provisionpipeline` API call.
2. Run the provisioned pipeline by uploading the training data to the S3 assets bucket specified in the output of the CloudFormation stack of the pipeline.
3. After the pipeline stack is provisioned, you can monitor the deployment of the model monitor via the AWS CodePipeline instance link listed in the output of the pipeline's CloudFormation template.

You can use the following AWS CLI commands to monitor and manage the lifecycle of the of the monitoring schedule job: `describe-monitoring-schedule`, `list-monitoring-executions`, and `stop-monitoring-schedule`.

Note
You must deploy a real-time inference pipeline first, and then deploy a model monitor pipeline to monitor the deployed Amazon SageMaker ML model. You must specify the name of the deployed Amazon SageMaker endpoint in the data quality, model quality, model bias, or model explainability monitor's API call.

Template option 2: Multi-account deployment

Important
You must set up the recommended AWS Organizations structure and enable trusted access with AWS Organizations before deploying the solution's multi-account deployment template option into the orchestrator account.

the section called “Step 1. Launch the stack” (p. 20)

the section called “Step 2. Provision the pipeline and train or deploy the ML model” (p. 24)

the section called “Step 3. Provision the model monitor pipeline (optional)” (p. 25)

Step 1. Launch the stack

Important
This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy.
To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your template and deploy the solution. For more information, refer to the Collection of operational metrics (p. 42) section of this guide.

This automated AWS CloudFormation template deploys AWS MLOps Framework in the AWS Cloud. You must have your model artifact's file and a Dockerfile before launching the stack.

Note
You are responsible for the cost of the AWS services used while running this solution. For more details, visit to the Cost section in this guide, and refer to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and use the button below to launch the `mlops-workload-orchestrator-multi-account-framework.template` AWS CloudFormation template.

   You can also download the template as a starting point for your own implementation.

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

   Note
   This solution uses the AWS CodePipeline service, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS CodePipeline is available. For the most current availability by Region, refer to the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, refer to IAM and STS Limits in the *AWS Identity and Access Management User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Email address	*<Requires input>*	Specify an email to receive notifications about pipeline outcomes.
CodeCommit repository address	<*Optional input*>	Only provide value if you want to provision a pipeline from a CodeCommit repository. For example: `https://git-codecommit.us-east-1.amazonaws.com/v1/repos/`*`<repository-name>`*
Existing S3 bucket name	<*Optional input*>	Optionally, provide the name of an existing S3 assets bucket to be used as the assets bucket. If an existing bucket is not provided, the solution creates a new S3 bucket. **Note** If you use an existing S3 bucket for the bucket must meet the following requirements: 1) the bucket must be in the same Region as the MLOps Workload Orchestrator stack, 2) the bucket must allow other AWS accounts (development, staging, and production) to read objects from the bucket (for example, using bucket policy), and 3) versioning must be allowed on the bucket. We recommended blocking public access, enabling S3 server-side encryption, access logging, and secure transport (for example, HTTPS only bucket policy) on your existing S3 bucket.
Existing Amazon ECR repository's name	<*Optional input*>	Optionally, provide the name of an existing Amazon ECR repository name to be used for custom algorithms images. If you do not provide an existing repository, the solution creates a new Amazon ECR repository

Parameter	Default	Description
		and adds required permissions for others AWS accounts (development, staging, and production) to pull images from the repository. **Note** The Amazon ECR repository you provide must be in the same Region where the solution is deployed. You must grant permissions to other AWS accounts to access the repository using a resource policy.
Do you want to use Amazon SageMaker Model Registry?	`No`	By default, this value is `No`. You must provide the algorithm and model artifact location. If you would like to use Amazon SageMaker model registry, you must set the value to `Yes`, and provide the model version ARN in the API call. For more details, refer to API operations (p. 27). The solution expects that the model artifact is stored in the S3 assets bucket. If you use a different S3 bucket with model registry, you must grant read permissions for the dev, staging, and prod accounts.

Parameter	Default	Description
Do you want the solution to create an Amazon SageMaker model package group?	`No`	By default, this value is `No`. If you use Amazon SageMaker model registry, you can set this value to `Yes` to instruct the solution to create a model registry (for example, model package group), and grant access permissions for other AWS accounts, development, staging, and prod (if you choose the multi-account deployment option). Otherwise, you can use your own model registry created outside the solution. **Note** If you choose to use a model registry that was not created by this solution, you must set up access permissions for other accounts to access the model registry. For more information refer to to Deploy a Model Version from a Different Account in the *Amazon SageMaker Developer Guide*.
Do you want to allow detailed error message in the APIs response?	`Yes`	By default, this value is `Yes` . If allowed, the API's response returns a detailed message for any server-side error/exception. If you set this parameter to `No` , the API's response returns a general error message.
Are you using a delegated administrator account (AWS Organizations)?	`Yes`	By default, this value is `Yes`. The solution expects that the orchestrator account is an AWS Organizations delegated administrator account. This follows best practices to limit the access to the AWS Organizations management account. However, if you want to use the management account as the orchestrator account, you can change this value to `No`.
Development Organizational Unit Id	*`<Requires input>`*	The AWS Organizations unit id for the development account (for example, `o-a1ss2d3g4`).

Parameter	Default	Description
Development account Id	*<Requires input>*	The development account's number.
Staging Organizational Unit Id	*<Requires input>*	The AWS Organizations unit ID for the staging account.
Staging account Id	*<Requires input>*	The staging account's number.
Production Organizational Unit Id	*<Requires input>*	The AWS Organizations unit ID for the production account.
Production account Id	*<Requires input>*	The production account's number.

Note
To connect a Github or BitBucket code repository to this solution, launch the solution and use the process in the source stage of the pipeline to create GitHubSourceAction and BitBucketSourceAction.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation Console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately three minutes.

Note
In addition to the primary AWS Lambda function (`AWSMLOpsFrameworkPipelineOrchestration`), this solution includes the solution-helper Lambda function, which runs only during initial configuration or when resources are updated or deleted.

When you run this solution, you will notice both Lambda functions in the AWS Management Console. Only the `AWSMLOpsFrameworkPipelineOrchestration` function is regularly active. However, you must not delete the solution-helper function, as it is necessary to manage associated resources.

Step 2. Provision the pipeline and train or deploy the ML model

Use the following procedure to provision the pipeline and train or deploy your ML model. If you are using API provisioning, the body of the API call must have the information specified in API operations (p. 27). API endpoints require authentication with IAM. For more information, refer to the How do I enable IAM authentication for API Gateway APIs? Support topic, and the Signing AWS requests with Signature Version 4 topic in the *AWS General Reference Guide*.

Note
If you are using API provisioning to launch the stack, you must make a POST request to the API Gateway endpoint specified in the stack's output. The path will be structured as *<apigateway_endpoint>*/provisionpipeline.
If you are using Git provisioning to launch the stack, you must create a file named `mlops-config.json` and commit the file to the repository's main branch.

1. Monitor the progress of the pipeline by calling the *<apigateway_endpoint>*/pipelinestatus. The pipeline_id is displayed in the response of the initial /provisionpipeline API call.
2. Run the provisioned pipeline by uploading the model artifacts to the S3 bucket specified in the output of the CloudFormation stack of the pipeline.

When the pipeline provisioning is complete, you will receive another apigateway_endpoint as the inference endpoint of the deployed model.

Step 3. Provision the model monitor pipeline (optional)

Use the following procedure to provision the pipeline and deploy data quality, model quality, model bias, or model explainability monitor. If you use API provisioning, the body of the API call must have the information specified in API operations (p. 27).

> **Note**
> If you use API provisioning to launch the stack, you must make a POST request to the API Gateway endpoint specified in the stack's output. The path will be structured as *<apigateway_endpoint>*/provisionpipeline.
> If you are using Git provisioning to launch the stack, you must create a file named `mlops-config.json` and commit the file to the repository's main branch.

1. Monitor the progress of the pipeline by calling the *<apigateway_endpoint>*/pipelinestatus. The `pipeline_id` is displayed in the response of the initial `/provisionpipeline` API call.
2. Run the provisioned pipeline by uploading the training data to the S3 assets bucket specified in the output of the CloudFormation stack of the pipeline.
3. After the pipeline stack is provisioned, you can monitor the deployment of the model monitor via the AWS CodePipeline instance link listed in the output of the pipeline's CloudFormation template.

You can use the following AWS CLI commands to monitor and manage the lifecycle of the of the monitoring schedule job: `describe-monitoring-schedule`, `list-monitoring-executions`, and `stop-monitoring-schedule`.

> **Note**
> You must deploy a real-time inference pipeline first, and then deploy a model monitor pipeline to monitor the deployed SageMaker ML model. You must specify the name of the deployed Amazon SageMaker endpoint in the model monitor's API call.

Additional resources

AWS services	
• AWS CloudFormation	• AWS CodeBuild
• Amazon SageMaker	• AWS CodePipeline
• AWS Lambda	• Amazon S3
• Amazon ECR	• AWS Key Management Service
• Amazon API Gateway	• AWS Identity and Access Management

API operations

You can use the following API operations to control the solution's pipelines. The following is a description of all attributes, and examples of required attributes per pipeline type.

Template option 1: Single account deployment

The MLOps Workload Orchestrator solution's AWS API Gateway has two main API endpoints, `/provisionpipeline`, used to provision a pipeline, and `/pipelinestatus`, used to get the status of a provisioned pipeline.

- `/provisionpipeline`
  - Method: POST
  - Body:
    - `pipeline_type`: Type of the pipeline to provision. The solution currently supports `byom_realtime_builtin` (real-time inference with Amazon SageMaker built-in algorithms pipeline), `model_training_builtin` (model training using Amazon SageMaker training pipeline), `model_tuner_builtin` (Amazon hyperparameter tuning pipeline), `model_autopilot_training` (Amazon SageMaker autopilot pipeline), `byom_realtime_custom` (real-time inference with custom algorithms pipeline), `byom_batch_builtin`, (batch transform with built-in algorithms pipeline), `byom_batch_custom` (batch transform with custom algorithms pipeline), `byom_data_quality_monitor` pipeline (data quality monitor), `byom_model_quality_monitor` pipeline (model quality monitor), `byom_model_bias_monitor` pipeline (model bias monitor), `byom_model_explainability_monitor` pipeline (model explainability monitor), and `byom_image_builder` (custom algorithm Docker image builder pipeline).
    - `custom_algorithm_docker`: Path to a zip file inside the S3 assets bucket, containing the necessary files (for example, Dockerfile, assets, etc.) to create a Docker image that can be used by Amazon SageMaker to deploy a model trained using the custom algorithm. For more information, refer to Example Notebooks: Use Your Own Algorithm or Model in the *Amazon SageMaker Developer Guide*.
    - `custom_image_uri`: URI of a custom algorithm image in an Amazon ECR repository.
    - `ecr_repo_name`: Name of an Amazon ECR repository where the custom algorithm image, created by the `byom_image_builder` pipeline, will be stored.
    - `image_tag`: custom algorithm's image tag to assign to the created image using the `byom_image_builder` pipeline.
    - `model_framework`: Name of the built-in algorithm used to train the model.
    - `model_framework_version`: Version number of the built-in algorithm used to train the model.
    - `model_name`: Arbitrary model name for the deploying model. The solution uses this parameter to create an Amazon SageMaker model, endpoint configuration, and endpoint with extensions on model name, such as `<model_name>-endpoint-config` and `<model_name>-endpoint`. The `model_name` is also used in the name of the deployed AWS CloudFormation stack for all pipelines.
    - `model_artifact_location`: Path to a file in the S3 assets bucket containing the model artifact file (the output file after training a model).
    - `model_package_name`: Amazon SageMaker model package name (for example, `"arn:aws:sagemaker:<region>:<account_id>:model-package/<model_package_group_name>/<model_version>"`).
    - `baseline_data`: Path to a csv file in S3 assets bucket containing the data with feature names used for training the model (for data quality, model bias, and model explainability monitors), or

model predictions and ground truth labels (for model quality monitor), for example a csv file with the header `"prediction, probability, label"` for a `BinaryClassification` problem.

- `inference_instance`: Instance type for inference (real-time or batch). Refer to Amazon SageMaker Pricing for a complete list of machine learning instance types.
- `data_capture_location`: Path to a prefix in an S3 Bucket (including the bucket's name, for example *<bucket-name>*/*<prefix>*) to store the data captured by the real-time Amazon SageMaker inference endpoint.
- `batch_inference_data`: Path to a file in an S3 Bucket (including the bucket's name, for example *<bucket-name>*/*<path-to-file>*) containing the data for batch inference. This parameter is not required if your inference type is set to `real-time`.
- `batch_job_output_location`: Path to a prefix in an S3 bucket (including the bucket's name, for example *<bucket-name>*/*<prefix>*) to store the output of the batch transform job. This parameter is not required if your inference type is set to `real-time`.
- `instance_type`: Instance type used by the data baseline and model monitoring jobs.
- `instance_volume_size`: Size of the EC2 volume in GB to use for the baseline and monitoring job. The size must be enough to hold your training data and create the data baseline.
- `instance_count`: the number of EC2 instances used by the training job.
- `endpoint_name`: The name of the deployed Amazon SageMaker endpoint to monitor when deploying data and model quality monitor pipelines. Optionally, provide the `endpoint_name` when creating a real-time inference pipeline which will be used to name the created Amazon SageMaker endpoint. If you do not provide `endpoint_name`, it will be automatically generated.
- `baseline_job_output_location`: Path to a prefix in an S3 bucket (including the bucket's name, for example *<bucket-name>*/*<prefix>*) to store the output of the data baseline job.
- `monitoring_output_location`: Path to a prefix in an S3 bucket (including the bucket's name, for example *<bucket-name>*/*<prefix>*) to store the output of the monitoring job.
- `schedule_expression`: Cron job expression to run the monitoring job. For example, `cron(0 * ? * * *)` will run the monitoring job hourly, `cron(0 0 ? * * *)` daily, etc.
- `baseline_max_runtime_seconds`: Specifies the maximum time, in seconds, the baseline job is allowed to run. If the attribute is not provided, the job will run until it finishes.
- `monitor_max_runtime_seconds`: Specifies the maximum time, in seconds, the monitoring job is allowed to run. For data quality and model explainability monitors, the value can be up to 3300 seconds for an hourly schedule. For model quality and model bias hourly schedules, this can be up to 1800 seconds.
- `kms_key_arn`: Optional customer managed AWS Key Management Service (AWS KMS) key to encrypt captured data from the real-time Amazon SageMaker endpoint, output of batch transform and data baseline jobs, output of model monitor, and Amazon Elastic Compute Cloud (Amazon EC2) instance's volume used by Amazon SageMaker to run the solution's pipelines. This attribute may be included in the API calls of `byom_realtime_builtin`, `byom_realtime_custom`, `byom_batch_builtin`, `byom_batch_custom`, and `byom_`*<monitor-type>*`_monitor` pipelines.
- `baseline_inference_attribute`: Index or JSON path to locate predicted label(s) required for `Regression` or `MulticlassClassification` problems. The attribute is used by the model quality baseline. If `baseline_probability_attribute` and `probability_threshold_attribute` are provided, `baseline_inference_attribute` is not required for a `BinaryClassification` problem.
- `baseline_probability_attribute`: Index or JSON path to locate predicted label(s) required for `Regression` or `MulticlassClassification` problems. The attribute is used by the model quality baseline. If `baseline_probability_attribute` and `probability_threshold_attribute` are provided, `baseline_inference_attribute` is not required for a `BinaryClassification` problem.
- `baseline_ground_truth_attribute`: Index or JSON path to locate actual label(s). Used by the model quality baseline.

- `problem_type`: Type of Machine learning problem. Valid values are `"Regression"`, `"BinaryClassification"`, or `"MulticlassClassification"`. Used by the model quality, model bias, and model explainability monitoring schedules. It is an optional attribute for the `model_autopilot_training` pipeline. If not provided, the autopilot job will infer the problem type from the `target_attribute`. If provided, the `job_objective` attribute must be provided too.
- `job_objetive`: (optional) Metric to optimize, used by the `model_autopilot_training` pipeline. If provided, the `problem_type` must be provided. Valid values `"Accuracy"`, `"MSE"`, `"F1"`, `"F1macro"`, `"AUC"`.
- `job_name`: (optional) The name of the training job. If not provided, a name will be automatically generated by the solution. Used by all training pipelines. Note: The given name must be unique (no previous jobs created by the same name).
- `training_data`: The S3 file key/prefix of the training data in the solution's S3 assets bucket. This attribute is required by all training pipelines. Note: For `model_training_builtin` and `model_tuner_builtin` pipelines, the csv should not have a header. The target attribute should be the first column. For `model_autopilot_training` pipeline, the file should have a header.
- `validation_data`: (optional) The S3 file key/prefix of the training data in the solution's S3 assets bucket. This attribute is used by the `model_training_builtin` and `model_tuner_builtin` pipelines.
- `target_attribute`: Target attribute name in the training data. Required by the `model_autopilot_training` pipeline.
- `compression_type`: (optional) Compression type used with the training/validation data. Valid values "Gzip".
- `content_type`: (optional) The MIME type of the training data. Default: `"csv"`.
- `s3_data_type`: (optional) Training S3 data type. Valid values `"S3Prefix"`, `"ManifestFile"`, or `"AugmentedManifestFile"`. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines. Default: `"S3Prefix"`.
- `data_distribution`: (optional) Data distribution. Valid values `"FullyReplicated"` or `"ShardedByS3Key"`. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines. Default: `"FullyReplicated"`.
- `data_input_mode`: (optional) Training data input mode. Valid `"File"`, `"Pipe"`, `"FastFile"`. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines. Default: `"File"`.
- `data_record_wrapping`: (optional) Training data record wrapping, if any. Valid values `"RecordIO"`. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines.
- `attribute_names`: (optional) List of one or more attribute names to use that are found in a specified `AugmentedManifestFile` (if `s3_data_type = "AugmentedManifestFile"`). Used by the `model_training_builtin` and `model_tuner_builtin` pipelines.
- `job_output_location`: S3 prefix in the solution's S3 assets bucket, where the output of the training jobs will be saved.
- `job_max_candidates`: (optional) Maximum number of candidates to be tried by the autopilot job. Default: `10`.
- `max_runtime_per_job`: (optional) Maximum runtime in seconds the training job is allowed to run. Default: `86400`.
- `total_max_runtime`: (optional) Autopilot total runtime in seconds allowed for the job. Default: `2592000`.
- `generate_definition_only`: (optional) Generate candidate definitions only by the autopilot job. Used by the `model_autopilot_training` pipeline. Default: `"False"`.
- `encrypt_inner_traffic`: (optional) Encrypt inner-container traffic for the job. Used by training pipelines. Default: `"True"`.
- `use_spot_instances`: (optional) Use managed spot instances with the training job. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines. Default: `"True"`.

- `Max_wait_time_spot_instances`: (optional) Maximum wait time in seconds for Spot instances (required if `use_spot_instances = True`). Must be greater than `max_runtime_per_job`. Default: `172800`.
- `algo_hyperparamaters`: Amazon SageMaker built-in Algorithm hyperparameters provided as a json object. Used by the `model_training_builtin` and `model_tuner_builtin` pipelines. Example: {`"eval_metric": "auc"`, `"objective": "binary:logistic"`, `"num_round": 400`, `"rate_drop": 0.3`}.
- `tuner_configs`: `sagemaker.tuner.HyperparameterTuner` configs (`objective_metric_name`, `metric_definitions`, `strategy`, `objective_type`, `max_jobs`, `max_parallel_jobs`, `base_tuning_job_name=None`, `early_stopping_type`) provided as a json object. Required by the `model_tuner_builtin` pipeline.

  Note: Some have default values and are not required to be specified. Example: {`"early_stopping_type": "Auto"`, `"objective_metric_name": "validation:auc"`, `"max_jobs": 10`, `"max_parallel_jobs": 2`}.
- `hyperparamaters_ranges`: Algorithm hyperparameters range used by the `Hyperparameters` job provided as a json object, where the key is hyperparameter name, and the value is list with the first item the type (`"continuous"|"integer"|"categorical"`) and the second item is a list of [`min_value`, `max_value`] for `"continuous"|"integer"` and a list of values for `"categorical"`. Required by the `model_tuner_builtin` pipeline.

  Example: {`“min_child_weight”: [“continuous”,[0, 120]]`, `“max_depth”: [“integer”,[1, 15]]`, `“optimizer”: [“categorical”,[“sgd”, “Adam”]]`)}
- `monitor_inference_attribute`: Index or JSON path to locate predicted label(s). Required for `Regression` or `MulticlassClassification` problems, and not required for a `BinaryClassification` problem. Used by the model quality, model bias, and model explainability monitoring schedules.
- `monitor_probability_attribute`: Index or JSON path to locate probabilities. Used only with a `BinaryClassification` problem. Used by the model quality monitoring schedule.
- `probability_threshold_attribute`: Threshold to convert probabilities to binaries. Used by the model quality monitoring schedule, and only with a `BinaryClassification` problem.
- `monitor_ground_truth_input`: Used by the model quality and model bias monitoring schedules to locate the ground truth labels. The solution expects you to use `eventId` to label the captured data by the Amazon SageMaker endpoint. For more information, refer to the Amazon SageMaker developer guide on how to Ingest Ground Truth Labels and Merge Them with Predictions.
- `bias_config`: a JSON object representing the attributes of sagemaker.clarify.BiasConfig. Required only for model bias monitor pipeline.
- `model_predicted_label_config`: a JSON object representing the attributes of sagemaker.clarify.ModelPredictedLabelConfig. Required only for model bias monitor pipeline and `problem_type` is `BinaryClassification`, or `MulticlassClassification`.
- `shap_config`: a JSON object representing the attributes of sagemaker.clarify.SHAPConfig. Required only for model explainability monitor. For the `“baseline”` attribute, you can provide a list of lists or as s3 csv file’s key (representing features values to be used as the baseline dataset in the kernel SHAP algorithm). If a file key is provided, the file must be uploaded to the solution’s S3 assets bucket before making the API call.

- Required attributes per pipeline type (Amazon SageMaker model registry is not used):
  - Model training using Amazon SageMaker training job (with required attributes):

```
{
  "pipeline_type": "model_training_builtin",
  "model_name": "<my-model-name>",
  "model_framework": "xgboost",
  "model_framework_version": "1",
```

```
  "job_output_location": "<s3-prefix-in-assets-bucket>",
  "training_data": "<path/to/training_data.csv>",
  "validation_data": "<path/to/validation_data.csv>",
  "algo_hyperparamaters": "<algo-hyperparameters-json-object>"
}
```

- Model training using Amazon SageMaker hyperparameter tuning Job (with required attributes):

```
{

  "pipeline_type": "model_tuner_builtin",
  "model_name": "<my-model-name>",
  "model_framework": "xgboost",
  "model_framework_version": "1",
  "job_output_location": "<s3-prefix-in-assets-bucket>",
  "training_data": "<path/to/training_data.csv>",
  "validation_data": "<path/to/validation_data.csv>",
  "algo_hyperparamaters": "<algo-hyperparameters-json-object>",
  "tuner_configs": "<tuner-configs-json-object>",
  "hyperparamaters_ranges": "<hyperparamaters-ranges-json-object>"
}
```

- Model training using Amazon SageMaker autopilot job (with required attributes):

```
{

  "pipeline_type": "model_autopilot_training",
  "model_name": "<my-model-name>",
  "job_output_location": "<s3-prefix-in-assets-bucket>",
  "training_data": "<path/to/training_data.csv>",
  "target_attribute": "<target-attribute-name>"
}
```

- Real-time inference with a custom algorithm for a machine learning model:

```
{
  "pipeline_type": "byom_realtime_custom",
  "custom_image_uri": "<docker-image-uri-in-Amazon-ECR-repo>",
  "model_name": "<my-model-name>",
  "model_artifact_location": "<path/to/model.tar.gz>",
  "data_capture_location": "<bucket-name>/<prefix>",
  "inference_instance": "ml.m5.large",
  "endpoint_name": "<custom-endpoint-name>"
}
```

- Real-time inference with an Amazon SageMaker built-in model:

```
{
"pipeline_type": "byom_realtime_builtin",
  "model_framework": "xgboost",
  "model_framework_version": "1",
  "model_name": "<my-model-name>",
  "model_artifact_location": "<path/to/model.tar.gz>",
  "data_capture_location": "<bucket-name>/<prefix>",
  "inference_instance": "ml.m5.large",
  "endpoint_name": "<custom-endpoint-name>"
}
```

- Batch inference with a custom algorithm for a machine learning model:

```
{
  "pipeline_type": "byom_batch_custom",
```

```
  "custom_image_uri": "<docker-image-uri-in-Amazon-ECR-repo>",
  "model_name": "<my-model-name>",
  "model_artifact_location": "<path/to/model.tar.gz>",
  "inference_instance": "ml.m5.large",
  "batch_inference_data": "<bucket-name>/<prefix>/inference_data.csv",
  "batch_job_output_location": "<bucket-name>/<prefix>"
}
```

- Batch inference with an Amazon SageMaker built-in model:

```
{
  "pipeline_type": "byom_batch_builtin",
  "model_framework": "xgboost",
  "model_framework_version": "1",
  "model_name": "<my-model-name>",
  "model_artifact_location": "<path/to/model.tar.gz>",
  "inference_instance": "ml.m5.large",
  "batch_inference_data": "<bucket-name>/<prefix>/inference_data.csv",,
  "batch_job_output_location": "<bucket-name>/<prefix>"
}
```

- Data quality monitor pipeline:

```
{
  "pipeline_type": "byom_data_quality_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "<path/to/training_data_with_header.csv>",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location": "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression”: "cron(0 * ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "3300"
}
```

- Model Quality Monitor pipeline (BinaryClassification problem):

```
{

  "pipeline_type": "byom_model_quality_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "<path/to/baseline_dataset.csv>",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location": "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression": "cron(0 0 ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "1800",
  "baseline_inference_attribute": "prediction",
  "baseline_probability_attribute": "probability",
  "baseline_ground_truth_attribute": "label",
  "probability_threshold_attribute": "0.5",
  "problem_type": "BinaryClassification",
  "monitor_probability_attribute": "0",
  "monitor_ground_truth_input": "<bucket-name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"
}
```

- Model quality monitor pipeline (`Regression` or `MulticlassClassification` problem):

```
{

  "pipeline_type": "byom_model_quality_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "<path/to/baseline_data.csv>",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location": "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression": "cron(0 0 ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "1800",
  "baseline_inference_attribute": "prediction",
  "baseline_ground_truth_attribute": "label",
  "problem_type": "Regression",
  "monitor_inference_attribute": "0",
  "monitor_ground_truth_input": "<bucket-name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"
}
```

- Model bias monitor pipeline (`BinaryClassification` problem):

```
{
  "pipeline_type": "byom_model_bias_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "path/to/traing_data_with_header.csv",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location”: "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression": "cron(0 0 ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "1800",
  "probability_threshold_attribute": "0.5",
  "problem_type": "BinaryClassification",
  "monitor_probability_attribute": "0",
  "bias_config": {
                "label_values_or_threshold": "<value>",
                "facet_name": "<value>",
                "facet_values_or_threshold": "<value>"
   },
  "model_predicted_label_config":{"probability": 0},
  "monitor_ground_truth_input": "<bucket-name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"
}
```

- Model bias monitor pipeline (`Regression` problem):

```
{
  "pipeline_type": "byom_model_bias_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "<path/to/training_data_with_header.csv>",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location”: "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression": "cron(0 0 ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "1800",
```

```
  "problem_type": "Regression",
  "monitor_inference_attribute": "0",
  "bias_config": {
                "label_values_or_threshold": "<value>",
                "facet_name": "<value>",
                "facet_values_or_threshold": "<value>"
   },
  "monitor_ground_truth_input": "<bucket-name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"
}
```

- Model explainability monitor pipeline (`BinaryClassification` problem):

```
{
  "pipeline_type": "byom_model_explainability_monitor",
  "model_name": "<my-model-name>",
  "endpoint_name": "xgb-churn-prediction-endpoint",
  "baseline_data": "<path/to/training_data_with_header.csv>",
  "baseline_job_output_location": "<bucket-name>/<prefix>",
  "data_capture_location”: "<bucket-name>/<prefix>",
  "monitoring_output_location": "<bucket-name>/<prefix>",
  "schedule_expression": "cron(0 0 ? * * *)",
  "instance_type": "ml.m5.large",
  "instance_volume_size": "20",
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "1800",
  "probability_threshold_attribute": "0.5",
  "problem_type": "BinaryClassification",
  "monitor_probability_attribute": "0",
  "shap_config": {
                "baseline": "<path/to/shap_baseline_dataset.csv>",
                "num_samples": "<value>",
                "agg_method": "mean_abs|mean_sq|median"
  }
}
```

- Custom algorithm image builder pipeline:

```
{
  "pipeline_type": "byom_image_builder",
  "custom_algorithm_docker": "<path/to/custom_image.zip>",
  "ecr_repo_name": "<name-of-Amazon-ECR-repository>",
  "image_tag": "<image-tag>"
}
```

Required attributes per pipeline type when the Amazon SageMaker model registry is used. When the model registry is used, the following attributes must be modified:

- Real-time inference and batch pipelines with custom algorithms:
  - Remove `custom_image_uri` and `model_artifact_location`
  - Add `model_package_name`
- Real-time inference and batch pipelines with Amazon SageMaker built-in algorithms:
  - Remove `model_framework`, `model_framework_version`, and `model_artifact_location`
  - Add `model_package_name`

Expected responses of API requests to `/provisonpipeline`:

- If the pipeline is provisioned for the first time (that is, if no existing pipeline with the same name), the response is:

```
{
```

```
  "message": "success: stack creation started",
  "pipeline_id": "arn:aws:cloudformation:<region>:<account-id>:stack/<stack-id>"
}
```

- If the pipeline is already provisioned, the response is:

```
 {
  "message": "Pipeline <stack-name> is already provisioned. Updating template
 parameters.",
  "pipeline_id": "arn:aws:cloudformation:<region>:<account-id>:stack/<stack-id>"
}
```

- If the pipeline is already provisioned, the `pipeline_type` is `byom_image_builder`, and there are updates to be performed, the response is:

```
{
  "message": "Pipeline <stack-name> is being updated.",
"pipeline_id": " arn:aws:cloudformation:<region>:<account-id>:stack/<stack-id>"
}
```

- If the pipeline is already provisioned, the `pipeline_type` is `byom_image_builder`, and there are no updates to be performed, the response is:

```
{
  "message": "Pipeline <stack-name> is already provisioned. No updates are to be
 performed.
"pipeline_id": " arn:aws:cloudformation:<region>:<account-id>:stack/<stack-id>"
}
```

- `/pipelinestatus`
  - Method: POST
  - Body
    - Pipeline_id: The ARN of the created CloudFormation stack after provisioning a pipeline. (This information can be retrieved from `/provisionpipeline`.)
  - Example structure:

```
{
  "pipeline_id": "arn:aws:cloudformation:us-west-1:123456789123:stack/my-mlops-
pipeline/12abcdef-abcd-1234-ab12-abcdef123456"
}
```

- Expected responses of APIs requests to `/pipelinestatus`:
  - The returned response depends on the solution's option (single- or multi-account deployment). Example response for the single-account option:

```
{
  "pipelineName": "<pipeline-name>",
  "pipelineVersion": 1,
  "stageStates": [
    {
      "stageName": "Source",
      "inboundTransitionState": {
        "enabled": true
      },
      "actionStates": [
        {
```

```
          "actionName": "S3Source",
          "currentRevision": {
            "revisionId": "<version-id>"
          },
          "latestExecution": {
            "actionExecutionId": "<execution-id>",
            "status": "Succeeded",
            "summary": "Amazon S3 version id: "<id>",
            "lastStatusChange": "<timestamp>",
            "externalExecutionId": "<execution-id>"
          },
          "entityUrl": "https://console.aws.amazon.com/s3/home?region=<region>#"
        }
      ],
      "latestExecution": {
        "pipelineExecutionId": "<execution-id>",
        "status": "Succeeded"
      }
    },
    {
      "stageName": "DeployCloudFormation",
      "inboundTransitionState": {
        "enabled": true
      },
      "actionStates": [
        {
          "actionName": "deploy_stack",
          "latestExecution": {
            "actionExecutionId": "<execution-id>",
            "status": "Succeeded",
            "summary": "Stack <pipeline-name> was created.",
            "lastStatusChange": "<timestamp>",
            "externalExecutionId": "<stack-id>",
            "externalExecutionUrl": ""<stack-url>"
          },
          "entityUrl": "https://console.aws.amazon.com/cloudformation/home?
region=<region>#/"
        }
      ],
      "latestExecution": {
        "pipelineExecutionId": "<execution-id>",
        "status": "Succeeded"
      }
    }
  ],
  "created": "<timestamp>",
  "updated": "<timestamp>",
  "ResponseMetadata": {
    "RequestId": "<request-id>",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "<request-id>",
      "date": "<date>",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "<number>"
    },
    "RetryAttempts": 0
  }
}
```


You can use the following API method for inference of the deployed real-time inference pipeline. The AWS Gateway API URL can be found in the outputs of the pipeline's AWS CloudFormation stack.

- `/inference`
  - Method: POST

- Body
  - Payload: The data to be sent for inference.
  - ContentType: MIME content type for the payload.

```
{
  "payload": "1.0, 2.0, 3.2",
  "content_type": "text/csv"
}
```

- Expected responses of APIs requests to `/inference`:
  - The request returns a single prediction value, if one data point was in the request, and returns multiple prediction values (separated by a ","), if several data points were sent in the APIs request.

API responses with error messages:

- If an API request to any one of the solution's API endpoints results in an exception/error, the expected body of the API response is:

```
{
  "message": "<general error message>",
  "detailedMessage ": "<detailed error message>"
}
```

- The `detailedMessage` attribute in the body of the API response is only included if the solution was configured to allow detailed error messages. Refer to the template's parameters table for more details.

Template option 2: Multi-account deployment

The same API calls used for single account development are used for multi-account deployment, with the exception of the following changes:

- For training pipelines, the API calls to provision the pipelines are similar to the single-account deployment. All training pipelines are deployed in the delegated admin account, where the solution is deployed.

  For BYOM real-time built-in and custom pipelines, you must provide the `inference_instance` and `data_capture_location`, endpoint_name (optional), and `kms_key_arn (optional)` for the development, staging, and production deployments. For example:

  - Real-time inference with an Amazon SageMaker built-in model:

```
{
  "pipeline_type" : "byom_realtime_builtin",
  "model_framework": "xgboost",
  "model_framework_version": "1",
  "model_name": "<my-model-name>",
  "model_artifact_location: "<path/to/model.tar.gz>",
  "data_capture_location": {"dev":"<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": "<bucket-name>/<prefix>"},
  "inference_instance": {"dev":"ml.t3.2xlarge", "staging":"ml.m5.large",
 "prod":"ml.m5.4xlarge"}
  "endpoint_name": {"dev": "<dev-endpoint-name>",
                    "staging": "<staging-endpoint-name>",
                    "prod": "<prod-endpoint-name>"}
}
```

- For BYOM batch built-in and custom pipelines, you must provide the `batch_inference_data`,`inference_instance`, `batch_job_output_location`, and `kms_key_arn (optional)`for the development, staging, and production deployments. For example:
  - Batch transform with a custom algorithm:

```
{
  "pipeline_type" : "byom_batch_custom",
  "custom_image_uri": "<docker-image-uri-in-Amazon-ECR-repo>",
  "model_name": "<my-model-name>",
  "model_artifact_location": "<path/to/model.tar.gz>",
  "inference_instance": {"dev":"ml.t3.2xlarge",
  "staging":"ml.m5.large", "prod":"ml.m5.4xlarge"},
  "batch_inference_data": {"dev":"<bucket-name>/<prefix>/data.csv", "staging":
 "<bucket-name>/<prefix>/data.csv", "prod": "<bucket-name>/<prefix>/data.csv"},
  "batch_job_output_location": {"dev":"<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": "<bucket-name>/<prefix>"}
}
```

- For the model monitor pipeline, you should provide `instance_type` and `instance_volume_size`, `endpoint_name`, `date_capture_location`, `baseline_job_output_location` , `monitoring_output_location`, and `kms_key_arn (optional)`. The `kms_key_arn` must be the same key used for the real-time inference pipeline. Additionally, for Model Quality monitor pipeline, `monitor_ground_truth_input` is needed for each account. For example:
  - Data quality monitor pipeline:

```
{
  "pipeline_type": "byom_data_quality_monitor",
  "endpoint_name": {"dev": "<dev_endpoint_name>",
  "staging":"staging_endpoint_name", "prod":"<prod_endpoint_name>"},
  "training_data": "<path/to/traing_data_with_header.csv>",
  "baseline_job_output_location": {"dev": "<bucket-name>/<prefix>", "staging":
 "<bucket-name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "data_capture_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "monitoring_output_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": "<bucket-name>/<prefix>"},
  "schedule_expression": "cron(0 * ? * * *)",
  "instance_type": {"dev":"ml.t3.2xlarge", "staging":"ml.m5.large",
 "prod":"ml.m5.4xlarge"},
  "instance_volume_size": {"dev":"20", "staging":"20", "prod":"100"},
  "baseline_max_runtime_seconds": "3300"
  "monitor_max_runtime_seconds": "3300"
}
```

  - Model quality monitor pipeline:

```
{

  "pipeline_type": "byom_model_quality_monitor",
  "endpoint_name": {"dev": "<dev_endpoint_name>",
  "staging":"staging_endpoint_name", "prod": "<prod_endpoint_name>"},
  "baseline_data": "<path/to/baseline_data.csv>",
  "baseline_job_output_location": {"dev": "<bucket-name>/<prefix>", "staging":
 "<bucket-name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "data_capture_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "monitoring_output_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "schedule_expression": "cron(0 * ? * * *)",
  "instance_type": {"dev":"ml.t3.2xlarge", "staging":"ml.m5.large",
 "prod":"ml.m5.4xlarge"},
  "instance_volume_size": {"dev":"20", "staging":"20", "prod":"100"},
```

```
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "3300",
  "baseline_inference_attribute": "prediction",
  "baseline_ground_truth_attribute": “label”,
  "problem_type": "Regression",
  "monitor_inference_attribute": "0",
  "monitor_ground_truth_input": {"dev": "<dev-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>", "staging": "<staging-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>", "prod": "<prod-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"}

}
```

- Model bias monitor pipeline:

```
{
  "pipeline_type": "byom_model_bias_monitor",
  "endpoint_name": {"dev": "<dev_endpoint_name>",
  "staging":"staging_endpoint_name", "prod":"<prod_endpoint_name>"},
  "baseline_data": "path/to/training_data_with_header.csv",
  "baseline_job_output_location": {"dev": "<bucket-name>/<prefix>", "staging":
 "<bucket-name>/<prefix>", "prod": “<bucket-name>/<prefix>”},
  "data_capture_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "monitoring_output_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "schedule_expression": "cron(0 * ? * * *)",
  "instance_type": {"dev":"ml.t3.2xlarge", "staging":"ml.m5.large",
 "prod":"ml.m5.4xlarge"},
  "instance_volume_size": {"dev":"20", "staging":"20", "prod":"100"},
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "3300",
  "problem_type": "Regression",
  "monitor_inference_attribute": "0",
  "bias_config": {
                "label_values_or_threshold": "<value>",
                "facet_name": "<value>",
                "facet_values_or_threshold": "<value>"
   },
  "monitor_ground_truth_input": {"dev": "<dev-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>", "staging": "<staging-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>", "prod": "<prod-bucket-
name>/<prefix>/<yyyy>/<mm>/<dd>/<hh>"}
}
```

- Model explainability monitor pipeline:

```
{
  "pipeline_type": "byom_model_explainability_monitor",
  "endpoint_name": {"dev": "<dev_endpoint_name>",
  "staging":"<staging_endpoint_name>", "prod":"<prod_endpoint_name>"},
  "baseline_data": "path/to/training_data_with_header.csv",
  "baseline_job_output_location": {"dev": "<bucket-name>/<prefix>", "staging":
 "<bucket-name>/<prefix>", "prod": “<bucket-name>/<prefix>”},
  "data_capture_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "monitoring_output_location": {"dev": "<bucket-name>/<prefix>", "staging": "<bucket-
name>/<prefix>", "prod": “<bucket-name>/<prefix>"},
  "schedule_expression": "cron(0 * ? * * *)",
  "instance_type": {"dev":"ml.t3.2xlarge", "staging":"ml.m5.large",
 "prod":"ml.m5.4xlarge"},
  "instance_volume_size": {"dev":"20", "staging":"20", "prod":"100"},
  "baseline_max_runtime_seconds": "3300",
  "monitor_max_runtime_seconds": "3300",
```

```
  "problem_type": "Regression",
  "monitor_inference_attribute": "0",
  "shap_config": {
                "baseline": "<path/to/shap_baseline_dataset.csv>",
                "num_samples": "<value>",
                "agg_method": "mean_abs|mean_sq|median"
  }
}
```

When the model registry is used, the following attributes must be modified:

- Real-time inference and batch pipelines with custom algorithms:
  - Remove `custom_image_uri` and `model_artifact_location`
  - Add `model_package_name`
- Real-time inference and batch pipelines with Amazon SageMaker built-in algorithms:
  - Remove `model_framework`, `model_framework_version`, and `model_artifact_location`
  - Add `model_package_name`

Uninstall the solution

To uninstall this solution, you must delete the AWS CloudFormation stack and any other stacks that were created as a result of the MLOps Workload Orchestrator. Because some AWS CloudFormation stacks use IAM roles created by previous stacks, you must delete AWS CloudFormation stacks in the reverse order they were created (delete the most recent stack first, wait for the stack deletion to be completed, and then delete the next stack).

Note
You must first delete any deployed model monitoring pipelines for a specific endpoint in order to delete that endpoint and its real-time inference pipeline.

The solution does **not** automatically delete the S3 assets bucket, Amazon SageMaker Model Registry, or Amazon Elastic Container Registry (ECR) repository. You must manually delete the retained resources.

It is recommended that you use tags to ensure that all resources associated with MLOps Workload Orchestrator are deleted. For example, all resources created by the CloudFormation should have the same tag. Then you can use Resources Groups & Tag Editor to confirm that all resources with the specified tag are deleted.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. Select this solution's installation stack.
3. Choose **Delete**.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Note
When using the multi-account deployment option, deleting the AWS CloudFormation stacks created in the orchestrator account will not automatically delete the stacks deployed in the dev, staging, and prod accounts. You must manually delete the stacks from within those accounts.

Collection of operational metrics

This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how you use this solution and related services and products. When allowed, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Version** - The solution version
- **Unique ID (UUID)** - Randomly generated, unique identifier for each MLOps Workload Orchestrator deployment
- **Timestamp** - Data-collection timestamp
- **gitSelected** - Whether or not an AWS CodeCommit repository is provided
- **Region** - The AWS Region where the solution was deployed
- **IsMultiAccount** - Which template option was deployed (multi-account or single account)
- **IsDelegatedAccount** - Whether an AWS Organization delegated administrator account, or a management account, is used to deploy the solution's multi-account deployment option
- **UseModelRegistry** - Whether Amazon SageMaker model registry is used or not

AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy. To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. Download the AWS CloudFormation template to your local hard drive.
2. Open the CloudFormation template with a text editor.
3. Modify the AWS CloudFormation template mapping section from:

```
"Send": {
  "AnonymousUsage" : { "Data" : "Yes" }
},
```

   to

```
"Send": {
  "AnonymousUsage" : { "Data" : "No" }
},
```

4. Sign in to the AWS CloudFormation console.
5. Select **Create stack**.
6. On the **Create stack** page, **Specify template** section, select **Upload a template file**.
7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.
8. Choose Next and follow the steps in Launch the stack (p. 15) in the Automated deployment section of this guide.

Source code

Visit the GitHub repository to download the templates and scripts for this solution, and to share your customizations with others.

Revisions

Date	Change
November 2020	Initial release
January 2021	Release v1.1.0: Model monitor pipeline to monitor the quality of deployed machine learning models. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
March 2021	Release v1.1.1: Updated the Amazon ECR scan on push property and repository names. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
May 2021	Release v.1.2.0: Added an option for multi-account deployments, and added the Custom Algorithm Image Builder pipeline. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
June 2021	Release v.1.3.0: Added the option to use Amazon SageMaker Model Registry, and the option to use AWS Organizations delegated administrator account (default option) to orchestrate the deployment of Machine Learning (ML) workloads across the AWS Organizations accounts. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
September 2021	Release v1.4.0: Added Amazon SageMaker Model Quality monitor pipeline to monitor the performance of a deployed model by comparing the predictions that the model makes with the actual ground truth labels that the model attempts to predict. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
December 2021	Release v1.4.1: Added documentation about how customers can integrate custom blueprints into the solution. Added a configurable flag to start/stop server-side error propagation. Updated APIs responses. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
January 2022	Release v1.5.0: The solution name was changed from “AWS MLOps Framework” to “MLOps Workload Orchestrator”. Added Amazon SageMaker Model Bias monitor pipeline to monitor predictions for bias on a regular basis, and generate alerts if bias beyond a certain

Date	Change
	threshold is detected. Added Amazon SageMaker Explainability monitor to monitor predictions for feature attribution drift on a regular basis. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.
May 2022	Release v2.0.0: Added three training pipelines to train ML models using Amazon SageMaker built-in algorithms and training job, hyperparameter tuning job, and autopilot job. Added Amazon EventBridge rules to monitor the state change of the training jobs. For more information about the changes, refer to the CHANGELOG.md file in the GitHub repository.

Contributors

The following individuals contributed to this document:

- Tarek Abdunabi
- Mohsen Ansari
- Zain Kabani
- Dylan Tong

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

MLOps Workload Orchestrator is licensed under the terms of the of the Apache License Version 2.0 available at The Apache Software Foundation.

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.