AWS MLOps Framework
Implementation Guide
# Table of Contents

Home ................................................................................................................................. 1
Cost ........................................................................................................................................ 3
   Example cost table .......................................................................................................... 3
Architecture overview .......................................................................................................... 5
   Template option 1: Single account deployment .............................................................. 5
   Template option 2: Multi-account deployment .............................................................. 6
   Shared resources and data between accounts ............................................................... 8
   Pipeline descriptions ....................................................................................................... 8
Design considerations ......................................................................................................... 11
   Bring Your Own Model pipeline .................................................................................. 11
   Custom pipelines ............................................................................................................ 11
      Provision pipeline ........................................................................................................ 11
      Get pipeline status ...................................................................................................... 12
   Regional deployments .................................................................................................... 12
AWS CloudFormation templates .......................................................................................... 13
Automated deployment ....................................................................................................... 14
   Prerequisites .................................................................................................................. 14
   Deployment overview ..................................................................................................... 14
      Template option 1: Single account deployment ......................................................... 14
      Step 1. Launch the stack ............................................................................................. 14
      Step 2. Provision the pipeline and deploy the ML model ........................................... 17
      Step 3. Provision the model monitor pipeline (optional) ............................................. 17
      Template option 2: Multi-account deployment ......................................................... 18
      Step 1. Launch the stack ............................................................................................. 18
      Step 2. Provision the pipeline and deploy the ML model ........................................... 17
      Step 3. Provision the model monitor pipeline (optional) ............................................. 23
Security ................................................................................................................................ 24
   IAM roles ........................................................................................................................ 24
   AWS Key Management Service (KMS) Keys .................................................................. 24
Additional resources .......................................................................................................... 25
API operations .................................................................................................................... 26
   Template option 1: Single account deployment .............................................................. 26
   Template option 2: Multi-account deployment .............................................................. 32
Uninstall the solution .......................................................................................................... 35
   Using the AWS Management Console ........................................................................ 35
   Using AWS Command Line Interface ......................................................................... 35
Collection of operational metrics ....................................................................................... 36
Source code ........................................................................................................................ 37
Revisions ............................................................................................................................. 38
Contributors ....................................................................................................................... 39
Notices ................................................................................................................................. 40
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. 38): September 2021)

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 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:

- Initiates a pre-configured pipeline through an API call or a Git repository
- Automatically deploys a trained model and provides an inference endpoint
- Monitors deployed machine learning models and detects any deviation in their data or model quality
- Supports running your own integration tests to ensure that the deployed model meets expectations
- Allows for provisioning multiple environments to support your ML model's life cycle
- Multi-account support for bring-your-own-model (BYOM) and Model Monitor pipelines
- Allows building and registering Docker images for custom algorithms that can be used for model deployment on an Amazon SageMaker endpoint
- The option to use Amazon SageMaker Model Registry to deploy versioned models.
- Notifies users of the pipeline outcome though SMS or email

The AWS MLOps Framework solution currently offers six pipelines:

- Two BYOM Real-time Inference pipelines for ML models trained using both Amazon SageMaker built-in algorithms and custom algorithms.
• Two BYOM Batch Transform pipelines for ML models trained using both Amazon 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.
• Two 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 or model quality.

In order 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 AWS MLOps Framework 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. At the date of publication, the cost for running this solution with the default settings in the US East (N. Virginia) Region is approximately $257.07 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.

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Dimensions</th>
<th>Cost per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon API Gateway</td>
<td>333 million requests (pipelines provisioning and real-time inference requests)</td>
<td>$3.50</td>
</tr>
<tr>
<td>AWS Lambda</td>
<td>Requests cost: 333 million requests x $0.20 (per 1 million requests)</td>
<td>$66.60</td>
</tr>
<tr>
<td></td>
<td>Compute cost: 333,000,000 (executions) x 128/1024 (GB) x 0.1 seconds (execution duration) x $0.00001667 (per GB-s)</td>
<td>$69.39</td>
</tr>
<tr>
<td>Amazon SageMaker Hosting: Real-Time Inference</td>
<td>1 instance (ml.m5.large) x $0.115/hour x 24 (hours) x 31 (days)</td>
<td>$85.56</td>
</tr>
<tr>
<td>Amazon SageMaker Baseline Jobs</td>
<td>1 instance (ml.m5.large) x $0.115/hour x 10/60 (hours, job duration) x 2 (jobs per month)</td>
<td>$0.04</td>
</tr>
<tr>
<td>Amazon SageMaker Model Monitor</td>
<td>1 instance (ml.m5.large) x $0.115/hour x 2 (jobs per day) x 10/60 (hours, job duration) x 31 (days)</td>
<td>$1.19</td>
</tr>
<tr>
<td>Amazon SageMaker Batch Transform</td>
<td>1 instance (ml.m5.large) x $0.115 (per hour) x 2 (hours, job duration) x 30 (days)</td>
<td>$6.90</td>
</tr>
<tr>
<td>AWS service</td>
<td>Dimensions</td>
<td>Cost per month</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Amazon S3</td>
<td>S3 Standard storage: 100GB x $0.023 (per GB)</td>
<td>$2.30</td>
</tr>
<tr>
<td></td>
<td>PUT requests: 10,000 requests x $0.000005 (per request)</td>
<td>$0.05</td>
</tr>
<tr>
<td></td>
<td>GET requests: 10,000 requests x $0.0000004 (per request)</td>
<td>$0.004</td>
</tr>
<tr>
<td>AWS CodePipeline</td>
<td>20 active pipelines x $1.00 (per month)</td>
<td>$20.00</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>Storage: 10 GB x $0.10 (per GB)</td>
<td>$1.00</td>
</tr>
<tr>
<td></td>
<td>Data transfer: 10 GB x $0.02 (per GB)</td>
<td>$0.20</td>
</tr>
<tr>
<td>AWS CodeBuild</td>
<td>10 builds per month x 10 minutes (build duration) x $0.0034 (per minute)</td>
<td>$0.34</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$257.07 per month</td>
</tr>
</tbody>
</table>
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 Amazon 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: AWS MLOps Framework solution architecture (single account)](image)

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 Amazon 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 Assets S3 bucket. If Amazon 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 CopePipeline’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 AWS MLOps Framework 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: AWS MLOps Framework 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 Assets S3 bucket in the AWS Orchestrator account. If Amazon 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 assets bucket, blueprint bucket, and Amazon ECR repository, and Amazon SageMaker Model Registry in the Orchestrator account. If the assets bucket, Amazon ECR repository, and Amazon 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 assets bucket and Amazon ECR repository, or are using an Amazon 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 Data/Model Quality 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**

**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 CodeBuild 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. As of September 2021, the solution supports Amazon SageMaker Data Quality and Model Quality monitoring. The data from Model Monitor reports can be used to set alerts for deviations in data and/or model quality. 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 monitoring, the solution creates a baseline from the dataset that was used to train the deployed model. For Model Quality monitoring, the baseline datasets 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. The baseline computes metrics and suggests constraints for the metrics and produces two files: `constraints.json` and `statistics.json`. The files are stored in the S3 bucket specified by `baseline_job_output_location` under the prefix `<baseline-job-name>/`.

3. 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 and Model Quality monitoring.
Design considerations

Bring Your Own Model pipeline

AWS MLOps Framework 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 pipelines

You can use a custom pipeline in this solution. Custom pipelines must be in an AWS CloudFormation template format, and must be stored in the Pipeline Blueprint Repository Amazon Simple Storage Storage (Amazon S3) bucket.

To implement a custom pipeline, you must have a firm understanding of the steps in your custom pipeline, and how your architecture implements those steps. You must also understand the input information your pipeline needs to deploy successfully. For example, if your pipeline has a step for training an ML model, your custom pipeline must know where the training data is located, and you must provide that information as input to the pipeline.

The orchestrator role in this solution provides you with two main controls that helps you manage your custom pipeline: Provision pipeline and Get pipeline status. These functionalities help you implement your custom pipeline. The following topics describe how you can connect each of these controls in your pipeline.

Provision pipeline

The solution orchestrator must ensure that your custom pipeline's CloudFormation template is stored in the Pipeline Blueprint Repository Amazon S3 bucket. The directory structure in the bucket must mirror the following format:

```
/
  <custom_pipeline_name>/
    <custom_pipeline_name>.yaml   # CloudFormation template
    lambdas/
      lambda_function_1/
        handler.py               # source code for a Lambda function
      lambda_function_2/
        handler.py
```

Replace `<custom_pipeline_name>` with the name of your pipeline. This name corresponds with pipeline_type in the API call or config file for provisioning a pipeline. For example, if your custom pipeline name is “mypipeline”, the value pipeline_type parameter should be “mypipeline”. This way, when the provision action is called either through API call or through AWS CodePipeline, the PipelineOrchestration Lambda function will create a CloudFormation stack using the template uploaded to blueprint-repository Amazon S3 bucket.
**Note**
Proper IAM permissions must be passed to mlopscloudformationrole so that when AWS CloudFormation assumes this role to provision the pipeline, it has access to provision all the necessary resources. For example, if your custom pipeline creates a Lambda function, mlopscloudformationrole must have lambda:CreateFunction permission to provision the Lambda function, and it must have lambda:DeleteFunction permission so that it can delete the Lambda function when the pipeline's CloudFormation stack is deleted.

**Get pipeline status**

Users can see the pipeline's status by calling the `/pipelinestatus` API. The pipelineOrchestration Lambda function calls its `pipeline_status()` method. When the pipeline is provisioning, the status returns the status of the CloudFormation stack. When the pipeline is running, there are multiple ways to show the status of a serverless architecture. One method for viewing the status is to publish events to a queue where the status method returns the latest event on the queue as the current status of the pipeline.

**Regional deployments**

This solution uses the AWS CodePipeline and Amazon SageMaker services, which are not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS CodePipeline and Amazon SageMaker are available. For the most current availability 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.

**aws-mlops-single-account-framework.template**: Use this template to launch the solution with the single account deployment option. The default configuration deploys two Amazon Simple Storage Service (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.

**aws-mlops-multi-account-framework.template**: Use this template to launch the solution with the multi-account deployment option. The default configuration deploys two Amazon Simple Storage Service (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. 14)

- Sign in to the AWS Management Console
- Review the CloudFormation template parameters
- Create and launch the stack

the section called “Step 2. Provision the pipeline and deploy the ML model” (p. 17)

the section called “Step 3. Provision the model monitor pipeline (optional)” (p. 17)

**Step 1. Launch the stack**

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 aws-mlops-single-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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email address</td>
<td>&lt;Requires input&gt;</td>
<td>Specify an email to receive notifications about pipeline outcomes.</td>
</tr>
<tr>
<td>CodeCommit repository address</td>
<td>&lt;Optional input&gt;</td>
<td>Only required if you want to provision a pipeline from a CodeCommit repository. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="https://git-codecommit.us-east-1.amazonaws.com/v1/repos/">https://git-codecommit.us-east-1.amazonaws.com/v1/repos/</a>&lt;repository-name&gt;</td>
</tr>
<tr>
<td>Existing S3 bucket name</td>
<td>&lt;Optional input&gt;</td>
<td>Optionally, provide the name of an existing S3 bucket to be used as the assets bucket. If an existing bucket is not provided, the solution creates a new S3 bucket. <strong>Note:</strong> 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 AWS MLOps Framework stack, 2) the bucket must allow reading/writing objects to/from the bucket, and 3) versioning must be allowed on the bucket.</td>
</tr>
</tbody>
</table>
### Step 1. Launch the stack

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
<td><strong>Default</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>We recommend blocking public access, enabling</td>
<td>&lt;Optional input&gt;</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Existing Amazon ECR</strong></td>
<td><strong>repository's name</strong></td>
<td>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. <strong>Note</strong>: The Amazon ECR repository must be in the same Region where the solution is deployed.</td>
</tr>
<tr>
<td>Do you want to use Amazon SageMaker Model Registry?</td>
<td>&lt;Requires input&gt;</td>
<td>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. The solution expects that the model artifact is stored in the assets S3 bucket.</td>
</tr>
<tr>
<td>Do you want the solution to create an Amazon SageMaker model package group?</td>
<td>&lt;Requires input&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 deploy the ML model**

Use the following procedure to provision the pipeline and 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. 26). 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 or Model Quality Monitor. If you use API provisioning, the body of the API call must have the information specified in API operations (p. 26).

**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 assets S3 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 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 or Model Quality 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. 18)

- Sign in to the AWS Management Console (orchestrator account)
- Review the CloudFormation template parameters
- Create and launch the stack

the section called “Step 2. Provision the pipeline and deploy the ML model” (p. 22)

the section called “Step 3. Provision the model monitor pipeline (optional)” (p. 23)

**Step 1. Launch the stack**

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 `aws-mlops-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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email address</td>
<td>&lt;Requires input&gt;</td>
<td>Specify an email to receive notifications about pipeline outcomes.</td>
</tr>
<tr>
<td>CodeCommit repository address</td>
<td>&lt;Optional input&gt;</td>
<td>Only required if you want to provision a pipeline from a CodeCommit repository. For example: [<a href="https://git-codecommit.us-east-1.amazonaws.com/v1/repos/">https://git-codecommit.us-east-1.amazonaws.com/v1/repos/</a>&lt;repository-name&gt;]</td>
</tr>
<tr>
<td>Existing S3 bucket name</td>
<td>&lt;Optional input&gt;</td>
<td>Optionally, provide the name of an existing S3 bucket to be used as the assets bucket. If an existing bucket is not provided, the solution creates a new S3 bucket. <strong>Note:</strong> 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 AWS MLOps Framework 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.</td>
</tr>
</tbody>
</table>
### Step 1. Launch the stack

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing Amazon ECR repository's name</td>
<td>&lt;Optional input&gt;</td>
<td>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 and adds required permissions for others AWS accounts (development, staging, and production) to pull images from the repository. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>Do you want to use Amazon SageMaker Model Registry?</td>
<td>&lt;Requires input&gt;</td>
<td>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 the API operations. The solution expects that the model artifact is stored in the assets S3 bucket. If you use a different S3 bucket with Model Registry, you must grant read permissions for the dev, staging, and prod accounts.</td>
</tr>
</tbody>
</table>
### AWS MLOps Framework Implementation Guide

**Step 1. Launch the stack**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you want the solution to create an Amazon SageMaker model package group?</td>
<td>&lt;Requires input&gt;</td>
<td>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 Deploy a Model Version from a Different Account in the Amazon SageMaker Developer Guide.</td>
</tr>
<tr>
<td>Are you using a delegated administrator account (AWS Organizations)?</td>
<td>&lt;Requires input&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>Development Organizational Unit Id</td>
<td>&lt;Requires input&gt;</td>
<td>The AWS Organizations unit id for the development account (for example, o-a1ss2d3g4).</td>
</tr>
<tr>
<td>Development account Id</td>
<td>&lt;Requires input&gt;</td>
<td>The development account’s number.</td>
</tr>
<tr>
<td>Staging Organizational Unit Id</td>
<td>&lt;Requires input&gt;</td>
<td>The AWS Organizations unit id for the staging account.</td>
</tr>
<tr>
<td>Staging account Id</td>
<td>&lt;Requires input&gt;</td>
<td>The staging account’s number.</td>
</tr>
<tr>
<td>Production Organizational Unit Id</td>
<td>&lt;Requires input&gt;</td>
<td>The AWS Organizations unit id for the production account.</td>
</tr>
<tr>
<td>Production account Id</td>
<td>&lt;Requires input&gt;</td>
<td>The production account’s number.</td>
</tr>
</tbody>
</table>
Step 2. Provision the pipeline and deploy the ML model

Use the following procedure to provision the pipeline and 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. 26). 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 <api_gateway_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 <api_gateway_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 api_gateway_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 or Model Quality Monitor. If you use API provisioning, the body of the API call must have the information specified in API operations (p. 26).

**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 assets S3 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 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 Model Monitor's API call.
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 Security Center.

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 AWS MLOps Framework 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.
Additional resources

AWS Services

- AWS CloudFormation
- Amazon SageMaker
- AWS Lambda
- Amazon ECR
- Amazon API Gateway
- AWS CodeBuild
- AWS CodePipeline
- Amazon Simple Storage Service
- AWS Key Management Service
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 AWS MLOps Framework 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), `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_model_monitor` (Model Monitor) and `byom_image_builder` (custom algorithm Docker image builder pipeline).
    - custom_algorithm_docker: Path to a zip file inside 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, and amazon-sagemaker-examples in this solution’s GitHub repository.
    - 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`.
    - model_artifact_location: Path to a file in Assets Bucket containing the model artifact file (the output file after training a model).
    - model_package_name: Amazon SageMaker model package name (e.g., "arn:aws:sagemaker:<region>:<account_id>:model-package/<model_package_group_name>/<model_version>").
    - baseline_data: Path to a csv file in Assets Bucket containing the data with feature names used for training the model. (for Data Quality monitor), or model predictions and ground truth labels (for Model Quality monitor), for example a csv file with the heard “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.
Template option 1: Single account deployment

- **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.

- **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. If the attribute is not provided, the job will run until it finishes. For Data Quality monitor, the value can be up to 3300 seconds for an hourly schedule. For Model Quality 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_model_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 monitoring schedule.

- **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 monitoring schedule.

- **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 monitoring schedule 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.

• Required attributes per pipeline type (Amazon SageMaker Model Registry is not used):

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

    ```json
    {
      "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:

    ```json
    {
      "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:

    ```json
    {
      "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:

    ```json
    {
      "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",
    }
    ```
Data Quality Monitor pipeline:

```json
{
    "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):

```json
{
    "pipeline_type": "byom_model_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 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):

```json
{
    "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_probability_attribute": "probability",
    "baseline_ground_truth_attribute": "label",
    "problem_type": "BinaryClassification",
    "monitor_probability_attribute": "0",
    "monitor_ground_truth_input": "<bucket-name>/<prefix>/ <yyyy>/<mm>/<dd>/<hh>
```
Template option 1: Single account deployment

```
"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>"
}
```

- 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 Amazon SageMaker Model Registry is used. When 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": "success: stack creation started",
  "pipeline_id": "Pipeline <pipeline-name> is already provisioned. Updating template parameters."
}
```

- `/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:

```json
{
  "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:

```json
{
  "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>"
          }
        }
      ]
    }
  ]
}
```
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.

```json
{
  "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.

## 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 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:
AWS MLOps Framework Implementation Guide
Template option 2: Multi-account deployment

```
{  
  "pipeline_type": "byom_realtime_builin",
  "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-url-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 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/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"
}

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>"
  }
}

When 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 the AWS MLOps Framework solution, you must delete the AWS CloudFormation stack and any other stacks that were created as a result of the AWS MLOps Framework. 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 AWS MLOps Framework 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 AWS MLOps Framework 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

Example data:

Running: \{t2.micro: 2\}, \{m3.large:2\}

Stopped: \{t2.large: 1\}, \{m3.xlarge:3\}

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 task.

Modify the AWS CloudFormation template mapping section from:

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

to

```json
"Send" : {
   "AnonymousUsage" : { "Data" : "No" }
},
```
Source code

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2020</td>
<td>Initial release</td>
</tr>
<tr>
<td>January 2021</td>
<td>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.</td>
</tr>
<tr>
<td>March 2021</td>
<td>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.</td>
</tr>
<tr>
<td>May 2021</td>
<td>Release v1.2.0: 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.</td>
</tr>
<tr>
<td>June 2021</td>
<td>Release v1.3.0: 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.</td>
</tr>
<tr>
<td>September 2021</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

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