AWS MLOps Framework

Implementation Guide

AWS MLOps Framework: Implementation Guide

Table of Contents

AWS MLOps Framework

AWS Solutions Implementation Guide

Publication date: **November 2020**

The AWS MLOps Framework solution deploys a robust pipeline that uses managed automation tools and machine learning (ML) services to simplify ML model development and production.

This implementation guide describes architectural considerations and configuration steps for deploying AWS MLOps Framework in the Amazon Web Services (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.

Overview

The machine learning (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.

The AWS MLOps Framework solution helps you streamline and enforce architecture best practices for machine learning (ML) model productionization. This solution is an extendable framework that provides a standard interface for managing ML pipelines for AWS ML services and third-party services. The solution's template allows customers to upload their trained models, configure the orchestration of the pipeline, trigger the start of the deployment process, move models through different stages of deployment, and monitor the successes and failures of the operations.

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
- Supports running your own integration tests to ensure that the deployed model meets expectations
- Allows you to provision multiple environments to support your ML model's life cycle
- Notifies users of the pipeline outcome via email

The solution deploys an AWS CloudFormation template that supports the incorporation of pre-trained models in its pipeline (also referred to as *bring your own model*), but the solution can be customized to include a training component within its architecture.

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 **$60.93 a month**. This price is calculated based on the assumption that an Amazon SageMaker real-time inference instance is running every hour in a month, which costs approximately $48.24 per month.

Prices are subject to change. For full details, see 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. This example assumes that 100 pipelines are provisioning and running per day, one inference requests is run per second,100 MB of storage per model size is used, and the solution uses 100GB of storage size in total.

The majority of the monthly cost is dependent on real-time inference in Amazon SageMaker. This estimate uses an ml.m5.large instance and assumes that it runs every hour of the month. 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	Total Cost
Amazon API Gateway	$4.55
AWS Lambda	$0.80
Amazon SageMaker	$48.24
Amazon S3	$2.30
AWS CodePipeline	$2.00
Amazon ECR	$1.44
AWS CodeBuild	$1.60
Total	~$60.93/month

Architecture overview

Deploying this solution **with the default parameters** builds the following environment in the AWS Cloud.

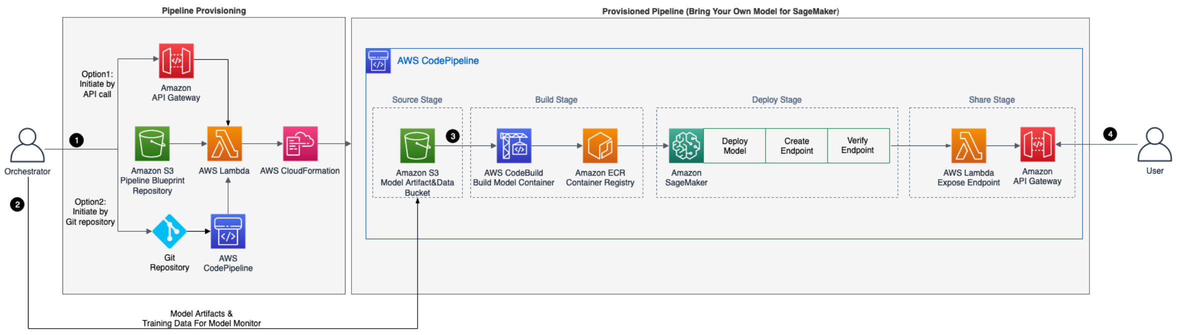
Pipeline Provisioning
Provisioned Pipeline (Bring Your Own Model for SageMaker)
Orchestrator
1
2
Option1: Initiate by API call
Amazon API Gateway
Amazon S3 Pipeline Blueprint Repository
AWS Lambda
AWS CloudFormation
Option2: Initiate by Git repository
Git Repository
AWS CodePipeline
AWS CodePipeline
Source Stage
Build Stage
Deploy Stage
Share Stage
Amazon S3 Model Artifact&Data Bucket
3
AWS CodeBuild Build Model Container
Amazon ECR Container Registry
Amazon SageMaker
Deploy Model
Create Endpoint
Verify Endpoint
AWS Lambda Expose Endpoint
Amazon API Gateway
4
User
Model Artifacts & Training Data For Model Monitor

Figure 1: AWS MLOps Framework solution architecture

The AWS CloudFormation template deploys a Pipeline Provisioning framework that provisions a machine learning pipeline (Bring Your Own Model for SageMaker). The template includes the AWS Lambda functions and AWS Identity and Access Management (IAM) roles necessary to set up your account, and it creates an Amazon Simple Storage Service (Amazon S3) bucket that contains the CloudFormation templates that set up the pipelines. The template also creates an Amazon API Gateway instance, an additional Lambda function, and an AWS CodePipeline instance.

Each provisioned pipeline includes four stages: **source**, **build**, **deploy**, and **share**.

- **Source**—AWS CodePipeline connects to an S3 bucket containing the provisioned model artifacts and a Dockerfile.
- **Build**—The pipeline uses the provisioned model artifacts (for example, a Python .pkl file) and the Dockerfile, to create a Docker image that is uploaded and stored in Amazon Elastic Container Registry (Amazon ECR).

> **Note**
> The *build* stage is only initiated when you specify that the model is a custom algorithm. If you are not using a custom algorithm, the *build* stage is skipped and an Amazon SageMaker container image is used instead. For more information around input parameters, refer to Appendix A.

- **Deploy**—An AWS Lambda function uses the Docker image to create a model and an endpoint on Amazon SageMaker.
- **Share**—An AWS Lambda function connects the Amazon SageMaker endpoint to Amazon API Gateway.

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

1. Orchestrator role (solution owner or DevOps engineer) launches the solution in their AWS account, then calls the API created in API Gateway to either provision the pipeline or commit the `mlops-config.json` file to the Git repository.
2. After provisioning, if your model is a custom algorithm not listed as an Amazon SageMaker built-in algorithm, the orchestrator uploads a zip file containing a Dockerfile and the necessary files for building an Amazon SageMaker compatible Docker image.
3. The orchestrator uploads the model artifact into the Assets bucket S3 bucket. The upload automatically initiates the pipeline.

   If the model is a custom algorithm, AWS CodeBuild uses the model artifact and the Dockerfile to build a Docker container image that is then stored in Amazon ECR.

   AWS Lambda function creates a model, an endpoint configuration, and an endpoint using the stored Docker image from Amazon ECR or a built-in image from Amazon SageMaker. Then, AWS CodePipeline initiates the share stage.

   In the share stage, an AWS Lambda function connects the created inference endpoint in SageMaker to API Gateway. This enables users to use their deployed model through the API created for inference.

   An Amazon SNS notification is sent to the email that was provided when launching the solution.
4. Users can test and use their deployed model using the API Gateway connected to their model in SageMaker.

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 execute 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/         # source code for a Lambda function
       handler.py               # used in the custom pipeline
     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 exmaple, 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.

Multi-account deployment

If you plan to deploy AWS MLOps Framework in a multi-account environment, you must deploy this solution in an AWS account that has permissions to provision resources in other accounts. The primary account (where the solution is deployed) and secondary accounts (where the pipelines are provisioned) must be in the same AWS Organizations. For more information, refer to the Managing the AWS accounts in your organization topic in the *AWS Organizations User Guide*.

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 Service Region Table.

AWS CloudFormation template

This solution uses AWS CloudFormation to automate the deployment of the AWS MLOps Framework solution in the AWS Cloud. It includes the following CloudFormation template, which you can download before deployment.

View Template

aws-mlops-framework.template: Use this template to launch the solution and all associated components. The default configuration deploys an Amazon Simple Storage Service (Amazon S3) bucket, an AWS Lambda function, an Amazon API Gateway API, an AWS CodePipeline project, and an AWS CodeBuild project. 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. Not required if using prebuild SageMaker Docker images
- A tool to call HTTP APIs (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.

Step 1. Launch the stack

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

Step 2. Provision the pipeline and deploy the ML model

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-framework.template` AWS CloudFormation template.

   Launch Solution

   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 Service Region Table.

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, see 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>*	Only required 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`

   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 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 Appendix A (p. 13).

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.

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 enable customers 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 services

- AWS CloudFormation
- Amazon SageMaker
- AWS Lambda
- Amazon ECR
- Amazon API Gateway
- AWS CodeBuild
- AWS CodePipeline
- Amazon S3

Appendix A: API Calls

You can use the following API methods to control the solution's pipelines. Note that you only need to provide the `batch_inference_data` parameter if the `inference_type` is `batch`.

- /provisionpipeline
  - Method: POST
  - Body:

```
{
  "pipeline_type" : "byom",
  "custom_model_container": "custom/model.zip",
  "model_framework": "xgboost",
  "model_framework_version": "1",
  "model_name": "my-model-name",
  "training_data": "training_data/data.csv",
  "inference_instance": "ml.m5.large",
  "inference_type": "realtime | batch",
  "batch_inference_data": "batch_inference/data.csv"
}
```

Note
If `inference_type` is set to `batch` the output of the batch transform is stored in the Assets S3 bucket (after Amazon SageMaker finishes the batch transform job).

- /pipelinestatus
  - Method: POST
  - Body

```
{
  “pipeline_id”: <pipeline_id>
}
```

You can use the following API method for inference of the deployed pipeline.

- /inference
  - Method: POST
  - Body

```
{
  "payload": <data for inference>,
  "ContentType": <MIME content type for payload>
}
```

Appendix B: Uninstall the solution

To uninstall the AWS MLOps Framework solution, you must delete the CloudFormation stack as well as any other CloudFormation stack that was created as a result of the AWS MLOps Framework. Deleting the CloudFormation stack may fail due to having retain policy on S3 buckets and IAM roles. You can choose to retain those resources and delete the CloudFormation stack after you delete the stack a second time. Or, you can manually delete the retained resources and then delete the CloudFormation stack.

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>
```

Appendix C: Collection of operational metrics

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. When enabled, the following information is collected and sent to AWS:

- **Solution ID:** The AWS solution identifier
- **Unique ID (UUID):** Randomly generated, unique identifier for each AWS MLOps Framework deployment
- **Timestamp:** Data-collection timestamp
- **Instance Data:** Count of the state and type of instances that are managed by the EC2 Scheduler in each AWS Region

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 as follows:

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

to

```
"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

Date	Change
November 2020	Initial release

Contributors

The following individuals contributed to this document:

- Mohsen Ansari
- Zain Kabani
- Dylan Tong

Notices