Journal of Artificial Intelligence and Machine Learning (JAIML)
Volume 3, Issue 2, July-December 2024, pp. 1-9, Article ID: JAIML_03_02_001
Available online at https://iaeme.com/Home/issue/JAIML?Volume=3&Issue=2
Journal ID: 5979-A561, DOI: https://doi.org/10.5281/zenodo.13897258

A CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT FRAMEWORK FOR SAGEMAKER REAL-TIME INFERENCE

Gautham Ram Rajendiran
USA

ABSTRACT

Real-time inference in a modern machine learning workflow requires robust deployment and monitoring to ensure models are delivering accurate and timely predictions. This paper elaborates on the implementation details of a CI/CD framework for deploying SageMaker real-time inference models by automating model packaging, deployment, and monitoring processes, integrating key approval steps that assure model performance and stakeholder involvement before production deployment. The workflow is designed to take full advantage of AWS Step Functions, SageMaker Model Registry, and other AWS services to make this transition from development to production as seamless as possible.

Keywords: Machine Learning Operations, Amazon Web Services, Real-Time Inference

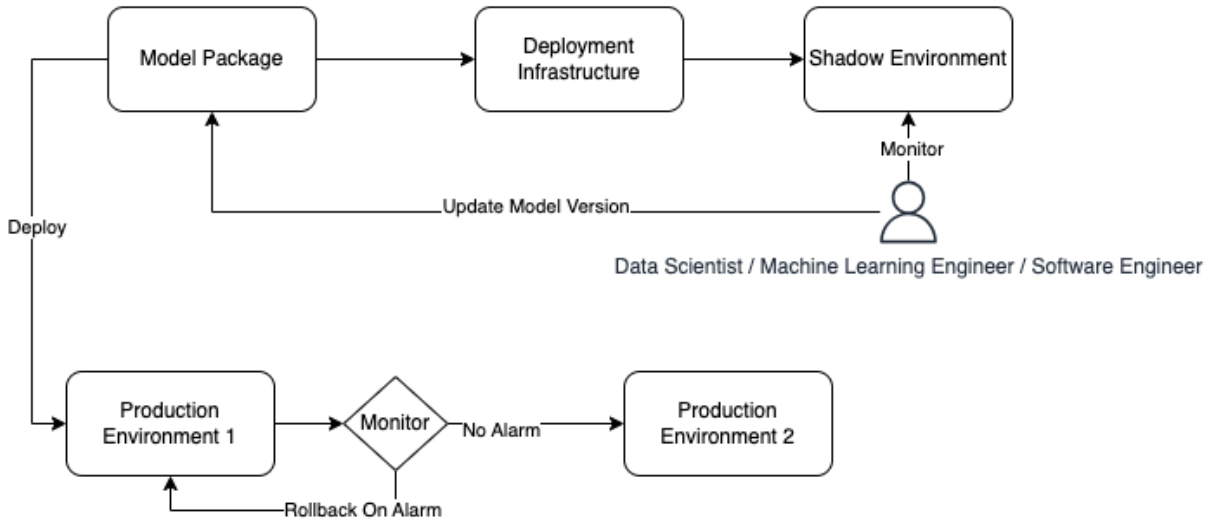
Cite this Article: Rajendiran, G. R. (2024). A Continuous Integration and Continuous Deployment Framework for Sagemaker Real-Time Inference. Journal of Artificial Intelligence and Machine Learning, 3(2), 1–9.
https://iaeme.com/Home/issue/JAIML?Volume=3&Issue=2

INTRODUCTION

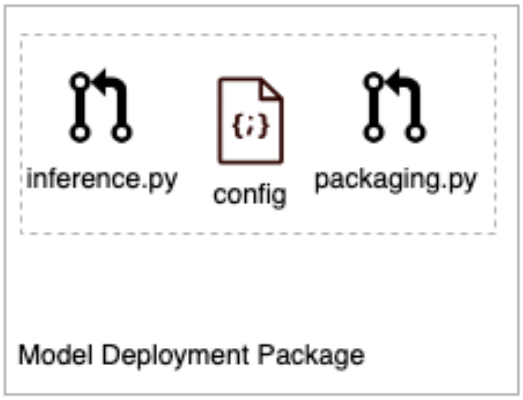
The models deployed for real-time inference use cases should be constantly updated with periodically re-trained models in order to account for data drift and performance enhancements. This creates the need for a structured CI/CD pipeline that will allow seamless model updates, thereby limiting manual intervention and production downtime. The framework ensures that automated checks, stakeholder approvals, and mechanisms for rollback are in place while facilitating fast iterative improvements. It also provides mechanisms to deploy new model changes in parallel to production models which can be used to measure the performance of the new model against the old model. This framework was implemented in a multi-national e-commerce company that supported real-time inference for 50+ use cases while reducing the time taken to iterate and deploy updates to machine learning models by 46%.

COMPONENTS OVERVIEW

This section provides a high level overview of all the components involved in the deployment automation framework. It also provides detail about implementation specifics of each of these components which will be useful for future works that aim to replicate this infrastructure. In the following section the paper elaborates on the flow of events by taking a case study as an example.

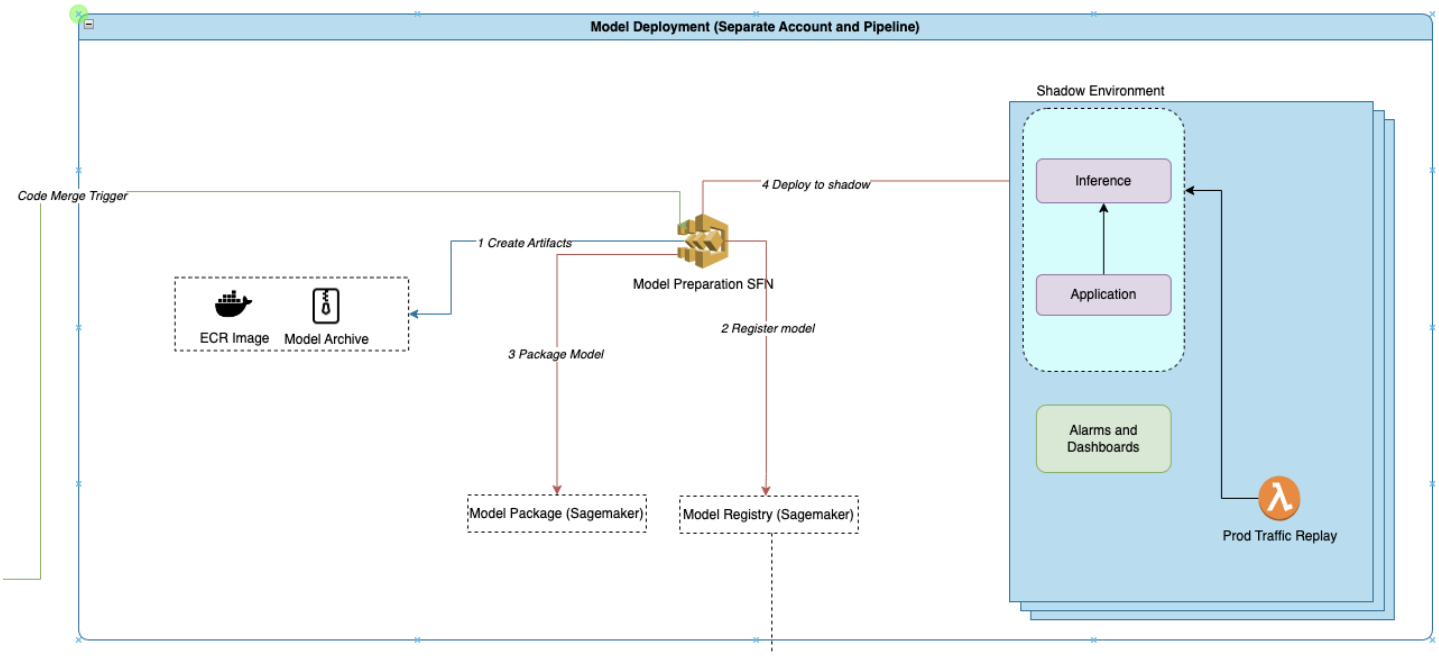


MODEL PACKAGE



A dedicated model package contains essential components to perform real time inference using Sagemaker. This includes the inference script, configuration files that contain environment specific information like instance type and ECR image to be used for a certain environment like production or shadow. It also contains a packaging script that has knowledge on how to create the tarball of the model which will be passed into Sagemaker Inference to instantiate the model.

DEPLOYMENT INFRASTRUCTURE



The deployment infrastructure is a combination of AWS Step functions, Sagemaker Model Registry, Sagemaker Model package and a shadow environment. These components work together to create a distributable model version that has been validated by having it serve production traffic for an extended period of time.

Prior to going into the implementation specifics, a note on the shadow environment needs to be made in order to properly understand the details. The shadow environment is a replica of the production environment and contains all infrastructure and code that has been deployed in production. It may or may not consume all traffic from the production environment depending on the use case. For example, this environment may need to consume 100% of the production traffic in order to estimate the performance of a certain instance type used for inference, and in some other cases it may need only 10% of the production traffic to test the performance metrics of a model. Such a lever is provided by the traffic replay component explained below.

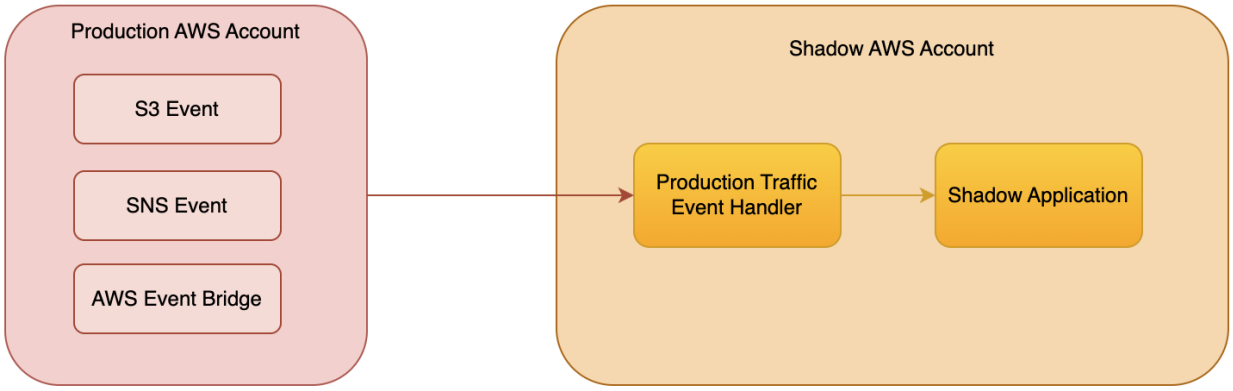
It uses AWS Step functions to orchestrate the packaging and deployment of models to shadow environments. The steps in detail are as follows:

1. Create Artifacts: This is the step where the pre-trained model file, the sagemaker inference script and other artifacts required to run the model are packaged into a tarball [1] in a way that is specified by Sagemaker here as shown in the documentation here [2]. It also creates a custom Docker image that will be deployed to AWS ECR, this is the image that will be used by Sagemaker to create the inference container which will serve requests to the model. The packaging script provided in the Model Deployment Package will contain implementation specifics on how to package the model.
2. Register Model: Using the artifacts created in the previous step, a Sagemaker Model [3] is created in Sagemaker Model Registry [4]. This provides mechanisms to automatically version control the model and perform comparisons across different model versions.
3. Package Model: The artifacts are also packaged into a Sagemaker Model Package [5] which allows the account to vend out the model to AWS accounts and other environments by providing a unique identifier called a ARN.

4. Deploy to Shadow: The model is deployed to the shadow environment in order to test it against live production traffic.

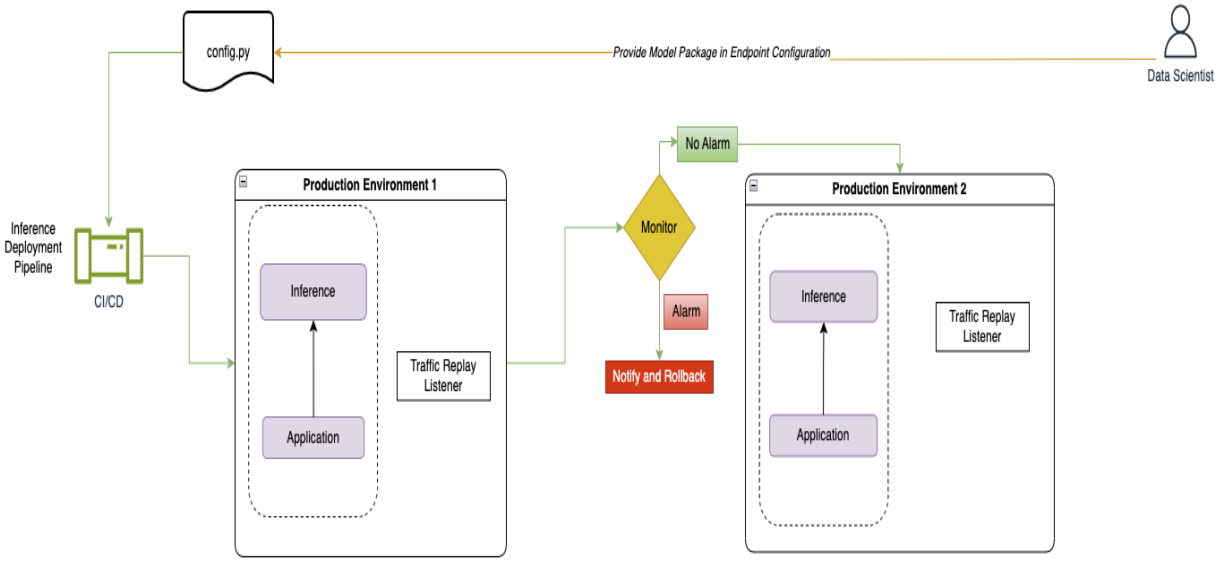
Once the model is deployed, it is monitored using pre-configured metrics in AWS Cloudwatch [6]. A data scientist or engineer monitors these metrics through dashboards and alarms in order to gauge the performance of the model against live production data. Once they are satisfied with the performance, the scientist makes note of the corresponding Model Package ARN of the model that is currently deployed in shadow. If they are not satisfied with the performance, another iteration of the deployment is performed with a different model.

TRAFFIC REPLAY



This component is used to route requests from the production environment into the shadow environment. It contains mechanisms to control the rate of traffic sent to shadow from production and acts as a middle-man for any transformations that need to be made to the routed requests in order to support testing the new model. It consists of an event handler that listens to all the events which are the entry points to the application in the production system, it then replicates these events in the shadow application while making transformations in the request as necessary.

ROLLBACK MONITOR



This is a monitor created using cloudwatch alarms which sends notification to a SNS topic on alarm. The code deployment pipeline listens to this SNS topic and reverts the most recent change deployed when this goes into the alarm stage. The deployments are orchestrated through AWS Code Commit [7] and AWS Code Deploy [8] which provide automated robust rollback mechanisms to revert the deployment in-case the newly deployed model under performs.

CASE STUDY: UPDATE A XGBOOST MODEL IN PRODUCTION

The idea of this case study is to explain infrastructure deployment, the addition of a trained model, and Inference Endpoint configuration configurations into the pipeline. It also explains how Step Functions are used to automate deployments and monitoring.

Step 1: Use CDK to Deploy Infrastructure

AWS CDK is employed to define and provision infrastructure components essential for the machine learning workflow:

- Amazon S3 for storing training data, model artifacts, and any intermediary files.
- Amazon SageMaker for managing model training and deploying inference endpoints.
- AWS Step Functions, CodeCommit, and CloudWatch Dashboards for automating the CI/CD workflow and monitoring model performance.

Step 2: Train the Model

The XGBoost model is trained using Amazon SageMaker's managed training services. After training, the model is serialized and saved for deployment.

- Training Script: The training script defines the dataset, hyperparameters, and evaluation metrics. The script runs in SageMaker to train the model.
- Model File: The trained model is saved in the form of a pickle file (.pkl) or PyTorch file (.pth), which will be used to create the deployment artifact.

Step 3: Prepare the Model for Deployment

The model artifact is converted into a tarball that includes the inference script and other configuration files required for deployment in SageMaker.

The folder structure might look like this:

```
model/
├── inference.py
├── model.pkl
└── Dockerfile
```

Step 4: Commit Changes to Git Repository

The model package (including the model location in S3, the inference script, and the endpoint configuration) is committed to a Git repository using AWS CodeCommit.

- The Git repository acts as the source of truth for all model and deployment artifacts.
- A pull request (PR) is created, which includes the trained model and the configurations needed for deployment.

Step 5: Trigger the CI/CD Pipeline

When a new commit is detected, AWS CodeDeploy triggers the pipeline. The Step Function orchestrates the steps to build the Docker image, package the model, and push it to Amazon ECR:

```
from aws_cdk import (
    aws_codebuild as codebuild,
    aws_codepipeline as codepipeline,
    aws_codepipeline_actions as actions,
)

# Define the pipeline
pipeline = codepipeline.Pipeline(self, "Pipeline")

# Source stage (from CodeCommit)
source_output = codepipeline.Artifact()
source_action = actions.CodeCommitSourceAction(
    repository=codecommit.Repository(self, "Repo", repository_name="XGBoostRepo"),
    branch="main",
    output=source_output
)
pipeline.add_stage(stage_name="Source", actions=[source_action])

# Build stage (to create model tarball and Docker image)
build_output = codepipeline.Artifact()
build_action = actions.CodeBuildAction(
    project=codebuild.Project(self, "BuildProject"),
    input=source_output,
    outputs=[build_output]
)
pipeline.add_stage(stage_name="Build", actions=[build_action])
```

Step 6: Register the Model in SageMaker

The Step Function registers the newly created model in SageMaker Model Registry for version control:

```
register_model_task = tasks.SageMakerRegisterModel(
    self, "RegisterModel",
    model_package_group_name="XGBoostModelPackageGroup",
    content_types=["text/csv"],
    inference_instances=["ml.m5.xlarge"],
    response_types=["text/csv"],
    transform_instances=["ml.m5.xlarge"],
    model_data=model_bucket.s3_url_for_object("model.tar.gz")
)
```

Step 7: Deploy to Shadow Environment

The Step Function deploys the model to a shadow environment, sending a small percentage of traffic to the new model while leaving the current production model intact:

```
shadow_deployment_task = tasks.SageMakerEndpointConfig(
  self, "ShadowDeployment",
  endpoint_config_name="XGBoostShadowEndpointConfig",
  production_variants=[
    tasks.SageMakerEndpointConfig.ProductionVariant(
      model_name="XGBoostModel",
      variant_name="ShadowVariant",
      initial_instance_count=1,
      instance_type="ml.m5.xlarge",
      initial_variant_weight=0.1
    )
  ]
)
```

Step 8: Monitor and Approve the Model

The data scientist monitors the model using a CloudWatch Dashboard. A sample CloudWatch alarm for monitoring prediction accuracy:

```
cloudwatch.Alarm(self, "PredictionAccuracyAlarm",
  metric=cloudwatch.Metric(
    namespace="AWS/SageMaker",
    metric_name="PredictionAccuracy",
    statistic="Average"
  ),
  threshold=0.95,
  evaluation_periods=5
)
```

Step 9: Update Production Endpoint

After the model has been validated in the shadow environment, the next step for the data scientist is to prepare the production deployment. The following steps are performed to deploy the shadow-verified model into production.

1. **Copy the SageMaker Model Package ARN:** Once the data scientist approves the model's performance in the shadow environment, they will copy the ARN (Amazon Resource Name) of the approved model package from the SageMaker Model Registry. This ARN uniquely identifies the versioned model that is to be deployed in production.
2. **Commit the ARN to the Deployment Package:** The data scientist will then update the deployment package by committing the new model package ARN. The deployment package contains configuration details, such as the inference endpoint configuration, which will point to this new version of the model.

Here's an example of how the data scientist might update the production deployment configuration:

```
production-inference-config:
        model_package_arn: "arn:aws:sagemaker:us-west-2:123456789012:model-package/xgboost-model-package/version"
    instance_type: "ml.m5.xlarge"
    initial_instance_count: 2
    variant_name: "ProductionVariant"
```

3. **Update Production Endpoint:** Once the deployment package has been updated and committed to the CodeCommit repository, the CI/CD pipeline is triggered. The pipeline automatically updates the SageMaker inference endpoint with the new model package version.

Step 10: Monitor Post-Deployment Performance

The deployment enters a monitoring phase. CloudWatch tracks critical metrics such as latency and accuracy. If alarms are triggered, the deployment is rolled back automatically.

Step 11: Rollback or Promote

The AWS Code Deployment pipeline can be configured in such a way that a deployment group or deployment can automatically roll back when a deployment fails or when a monitoring threshold is met. In this case, the last known good version of an application revision is deployed. It can also configure automatic rollbacks when the application is created or create or update a deployment group.

When a new deployment is created, one can also choose to override the automatic rollback configuration that was specified for the deployment group.

CONCLUSION

This CI/CD framework for SageMaker real-time inference provides a reliable, automated, and scalable solution for deploying machine learning models in production environments. This paper demonstrates the use of AWS Code Deploy to automate the deployment of a machine learning model to production and also shows ways to automate deployment rollbacks in-case of problems in performance. This method of continuous infrastructure and deployment of machine learning models was deployed in a large-scale eCommerce organization that had 50+ models running in production, with such a robust pipeline to support a streamlined deployment process. The aim of this paper is to expose this architecture for other engineers and scientists to use for a similar cloud-native setup on AWS, and reduce much of the time in starting fresh.

REFERENCES

[1] "TarBall - Debian Wiki," Debian. [Online]. Available: https://wiki.debian.org/TarBall.

[2] "Using XGBoost with SageMaker — Write an inference script," Amazon SageMaker Documentation. [Online]. Available: https://sagemaker.readthedocs.io/en/stable/frameworks/xgboost/using_xgboost.html#write-an-inference-script.

[3] "SageMaker Python SDK — Model," Amazon SageMaker Documentation. [Online]. Available: https://sagemaker.readthedocs.io/en/stable/api/inference/model.html.

[4] "Model Registry," Amazon SageMaker Documentation. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/model-registry.html.

[5] "Using a Model Package in SageMaker," Amazon SageMaker Documentation. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-mkt-model-pkg-model.html.

[6] "What is Amazon CloudWatch?" Amazon CloudWatch Documentation. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html "AWS CodeCommit — Source Control Service," Amazon Web Services. [Online]. Available: https://aws.amazon.com/codecommit/.

[7] "AWS CodeDeploy — User Guide," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html.

Citation: Rajendiran, G. R. (2024). A Continuous Integration and Continuous Deployment Framework for Sagemaker Real-Time Inference. Journal of Artificial Intelligence and Machine Learning, 3(2), 1–9

Abstract Link:
https://iaeme.com/Home/article_id/JAIML_03_02_001

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/JAIML/VOLUME_3_ISSUE_2/JAIML_03_02_001.pdf

✉ **editor@iaeme.com**