PRJ-MLS-043: End-to-End MLOps with SageMaker Pipelines

Certification: AWS Certified Machine Learning – Specialty
Domain: MLOps and Automation

---

1. Project Overview

This project demonstrates how to build a complete, end-to-end, automated MLOps workflow using **Amazon SageMaker Pipelines**. Manually managing the steps of a machine learning project (data preparation, training, evaluation, deployment) is slow, error-prone, and not scalable. SageMaker Pipelines is the first purpose-built, easy-to-use continuous integration and continuous delivery (CI/CD) service for machine learning. It allows you to define your entire ML workflow as a series of interconnected steps, creating a directed acyclic graph (DAG) that can be executed, tracked, and automated.

We will build a pipeline that automatically preprocesses data, trains a model, evaluates its performance, and, based on a conditional check of its accuracy, registers the model in the **SageMaker Model Registry**. This entire pipeline can be triggered manually, on a schedule, or automatically by a code change in a Git repository, enabling true CI/CD for machine learning.

Key Objectives

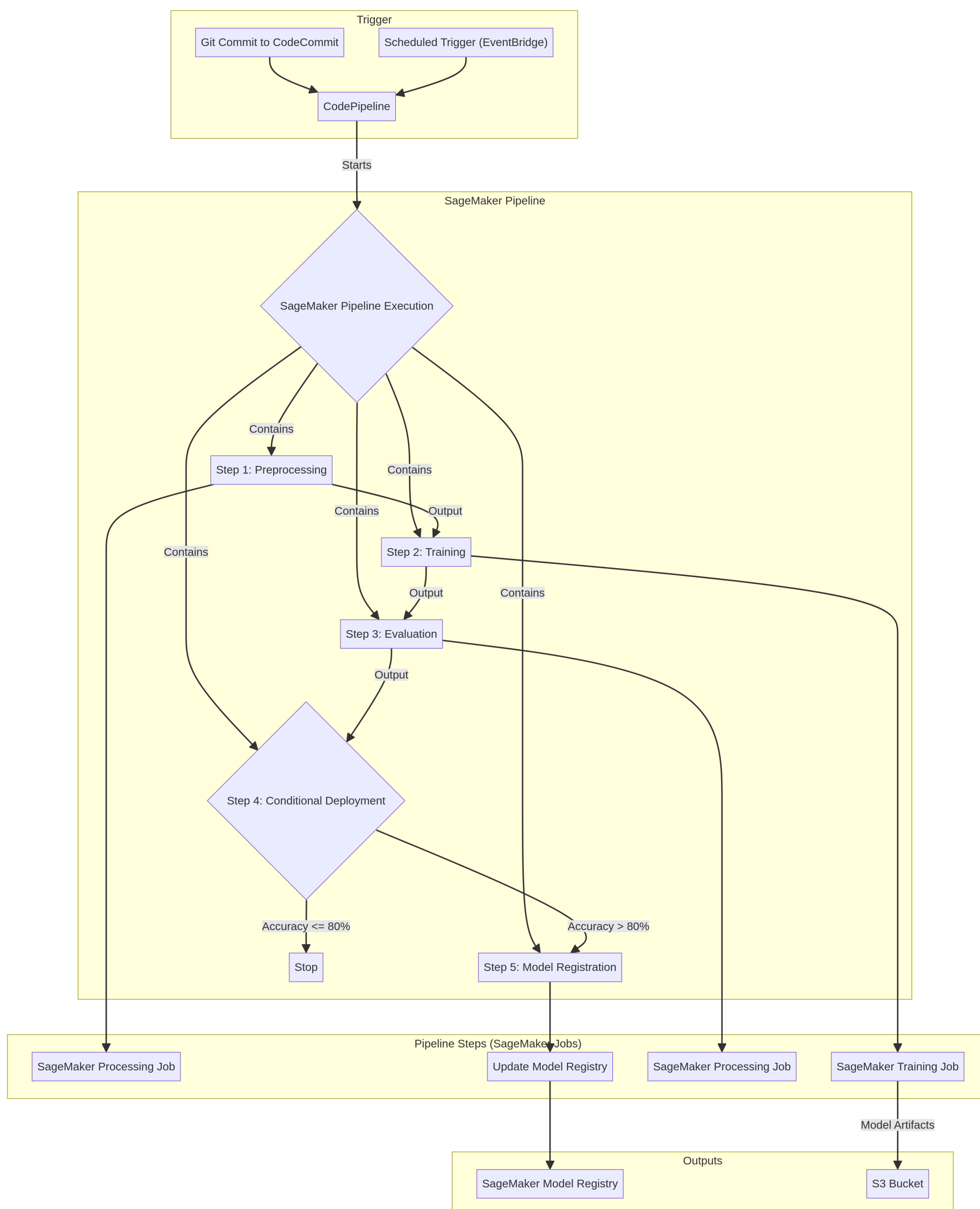
- Define a multi-step ML workflow using the SageMaker Pipelines SDK.
- Create distinct steps for data processing, model training, and model evaluation.
- Implement a **Condition Step** to make decisions based on model performance (e.g., only register the model if accuracy is above a certain threshold).
- Register the approved model in the SageMaker Model Registry for versioning and deployment.

- Integrate the SageMaker Pipeline with AWS developer tools like **CodePipeline** to trigger executions from a `git push`.

---

2. Architecture

The architecture centers on the SageMaker Pipeline DAG, which orchestrates various SageMaker jobs and can be integrated with CI/CD tools.



MLOps Workflow:

1. **Trigger:** The pipeline execution can be triggered in multiple ways:
    - **Manual:** A data scientist starts the pipeline from a SageMaker Studio notebook.
    - **Scheduled:** An **Amazon EventBridge** rule triggers the pipeline on a recurring schedule (e.g., weekly retraining on new data).
    - **CI/CD:** A `git push` to a specific branch in an **AWS CodeCommit** repository triggers **AWS CodePipeline**, which in turn starts the SageMaker Pipeline execution.
2. **SageMaker Pipeline Execution:**
    - A new execution of the pipeline DAG is created.
    - **Step 1: Preprocessing:** A `ProcessingStep` is initiated. This runs a script (e.g., scikit-learn) inside a **SageMaker Processing Job** to perform feature engineering and split the data into training and validation sets. The outputs are stored in S3.
    - **Step 2: Training:** A `TrainingStep` takes the preprocessed data as input and runs a **SageMaker Training Job** to train the model.
    - **Step 3: Evaluation:** Another `ProcessingStep` takes the trained model and the validation data. It runs an evaluation script to calculate performance metrics (e.g., accuracy, F1 score) and saves them as a JSON file.
    - **Step 4: Conditional Deployment:** A `ConditionStep` reads the accuracy from the evaluation report. It checks if the accuracy is greater than a predefined threshold (e.g., 80%).
    - **Step 5: Model Registration:** If the condition is met, a `ModelStep` is executed. This step takes the model artifacts and registers a new version in the **SageMaker Model Registry**. If the condition is not met, the pipeline branch terminates, and the model is not registered.
3. **Outputs:**
    - The primary output is a new, versioned, and approved model in the Model Registry, ready for deployment.

    - All intermediate artifacts (datasets, evaluation reports) are stored in S3 for lineage and auditing.

---

3. Prerequisites

- An AWS account with administrative permissions.
- A SageMaker Studio environment or a configured SageMaker Notebook instance.
- A dataset in S3 and corresponding Python scripts for preprocessing, training, and evaluation.

---

4. Step-by-Step Implementation Guide

This guide will be implemented in a SageMaker notebook using the SageMaker Python SDK.

Step 4.1: Define Pipeline Parameters

Parameters allow you to start pipeline executions with different values without changing the pipeline definition itself.

```
from sagemaker.workflow.parameters import ParameterInteger, ParameterString

processing_instance_count = ParameterInteger(name="ProcessingInstanceCount",
default_value=1)
input_data = ParameterString(name="InputData",
default_value=f"s3://{bucket}/my-input-data/")
```

Step 4.2: Define the Preprocessing Step

```
from sagemaker.sklearn.processing import SKLearnProcessor
from sagemaker.workflow.steps import ProcessingStep

sklearn_processor = SKLearnProcessor(
    framework_version="0.23-1",
    role=role,
    instance_type="ml.m5.large",
    instance_count=processing_instance_count
)

step_process = ProcessingStep(
    name="DataProcessing",
    processor=sklearn_processor,
    inputs=[sagemaker.processing.ProcessingInput(source=input_data,
destination="/opt/ml/processing/input")],
    outputs=[
        sagemaker.processing.ProcessingOutput(output_name="train",
source="/opt/ml/processing/train"),
        sagemaker.processing.ProcessingOutput(output_name="validation",
source="/opt/ml/processing/validation"),
    ],
    code="preprocess.py"
)
```

Step 4.3: Define the Training Step

```
from sagemaker.estimator import Estimator
from sagemaker.workflow.steps import TrainingStep

model_estimator = Estimator(
    image_uri=training_image, # e.g., SageMaker's built-in XGBoost image
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge"
)

step_train = TrainingStep(
    name="ModelTraining",
    estimator=model_estimator,
    inputs={
        "train": sagemaker.inputs.TrainingInput(

s3_data=step_process.properties.ProcessingOutputConfig.Outputs["train"].S3Outp
        )
    }
)
```

Step 4.4: Define the Evaluation and Condition Steps

```
from sagemaker.workflow.steps import ProcessingStep
from sagemaker.workflow.properties import PropertyFile

# Create an evaluation step that generates a report
eval_processor = SKLearnProcessor(...)
evaluation_report = PropertyFile(name="EvaluationReport",
output_name="evaluation", path="evaluation.json")
step_eval = ProcessingStep(
    name="ModelEvaluation",
    processor=eval_processor,
    inputs=[...], # Model from training step and validation data from
processing step
    outputs=[sagemaker.processing.ProcessingOutput(output_name="evaluation",
source="/opt/ml/processing/evaluation")],
    property_files=[evaluation_report],
    code="evaluate.py"
)

# Create a condition step based on the report
from sagemaker.workflow.conditions import ConditionGreaterThanOrEqualTo
from sagemaker.workflow.condition_step import ConditionStep

cond_gte = ConditionGreaterThanOrEqualTo(

left=sagemaker.workflow.functions.JsonGet(step_property=evaluation_report,
json_path="metrics.accuracy.value"),
    right=0.80 # Threshold
)

step_cond = ConditionStep(
    name="CheckAccuracy",
    conditions=[cond_gte],
    if_steps=[step_register_model], # Define this step next
    else_steps=[]
)
```

Step 4.5: Define the Model Registration Step

```
from sagemaker.workflow.steps import ModelStep

step_register_model = ModelStep(
    name="RegisterModel",
    estimator=model_estimator,
    model_data=step_train.properties.ModelArtifacts.S3ModelArtifacts,
    # ... other registration parameters ...
)
```

Step 4.6: Assemble and Execute the Pipeline

```
from sagemaker.workflow.pipeline import Pipeline

pipeline = Pipeline(
    name="MyMLPipeline",
    parameters=[processing_instance_count, input_data],
    steps=[step_process, step_train, step_eval, step_cond]
)

# Create/update and execute the pipeline
pipeline.upsert(role_arn=role)
pipeline.start()
```

---

5. CI/CD Integration with CodePipeline

1. **Source Stage:** Create a CodeCommit repository to store your pipeline definition code (`pipeline.py`), preprocessing script, etc.
2. **Build Stage:** Use AWS CodeBuild to run the `pipeline.py` script. This script will define and execute the `pipeline.upsert()` command, which creates or updates the SageMaker Pipeline definition.
3. **Deploy Stage:** Add an action to your CodePipeline that directly invokes the SageMaker Pipeline. You can specify the pipeline name, and CodePipeline will start a new execution every time the source code changes.

---

6. Cleanup

- **Delete the Pipeline:** In the SageMaker Studio console, you can delete the pipeline definition.
- **Delete Model Packages:** Go to the Model Registry and delete the model package versions created by the pipeline.
- **Clean up S3 Artifacts:** Delete the data from the S3 buckets used for pipeline inputs and outputs.
- **Delete CodePipeline:** Delete the CodePipeline and its associated resources (CodeCommit repo, CodeBuild project).

Business Context

The Problem

Organizations want to leverage AI/ML for business insights but lack the expertise to build secure, scalable ML systems. ML projects fail due to poor data quality, model drift, and lack of monitoring. Security and compliance requirements for ML systems are often overlooked.

The Solution

Production-ready ML system with automated data pipelines, model training, deployment, and monitoring. Implements MLOps best practices with version control, A/B testing, and automated retraining. Ensures data privacy and model security throughout the ML lifecycle.

Business Value

- **Business Intelligence:** AI-driven insights improve decision-making and outcomes
- **Operational Efficiency:** Automated ML workflows reduce manual data science effort by 60%
- **Model Reliability:** Continuous monitoring detects and corrects model drift automatically

- **Compliance Assurance:** Built-in controls for data privacy and model governance

Risk Mitigation

Protects sensitive data in ML pipelines, prevents biased or inaccurate models, ensures model explainability, and maintains compliance with AI regulations.

GRC Mapping

Compliance Frameworks

- **NIST AI RMF:** Trustworthy and responsible AI
- **ISO/IEC 23894:** AI risk management
- **NIST CSF:** PR.DS-2 (Data in transit), PR.DS-5 (Data leak protection)
- **Responsible AI Principles:** Fairness, accountability, transparency, ethics

Security Controls Implemented

- Data anonymization and pseudonymization
- Model explainability and interpretability
- Bias detection and mitigation
- Model versioning and rollback
- Automated model monitoring and alerting

Audit Evidence

- Model cards documenting intended use and limitations
- Bias and fairness evaluation reports
- Model performance metrics over time
- Data processing and transformation logs

Regulatory Alignment

- **GDPR:** Article 22 (Right to explanation), Article 25 (Privacy by design)

- **AI Act (EU):** Transparency and accountability requirements
- **CCPA:** Data privacy for ML training data
- **SOC 2:** CC6.1 (Access to sensitive data)