PRJ-MLE-006: Scalable Batch Inference Pipeline

Certification: AWS Certified Machine Learning Engineer – Associate
Domain: ML Infrastructure

---

1. Project Overview

This project focuses on building a cost-effective, scalable, and automated pipeline for batch inference using SageMaker Batch Transform. Unlike real-time endpoints, batch transform is ideal for processing large datasets offline, where predictions are not needed instantaneously. The pipeline will be triggered automatically when new data arrives in an S3 bucket, and the results will be stored back in S3, ready for analysis with Athena.

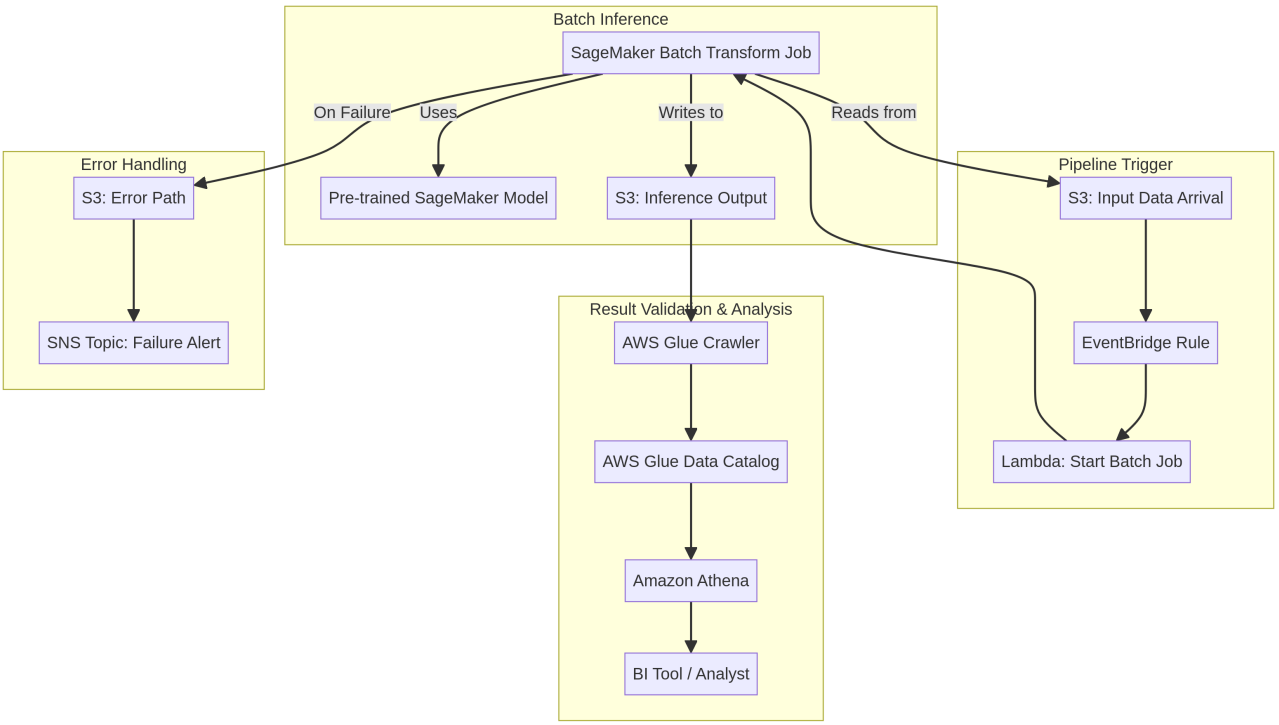
Key Objectives

- Train a machine learning model and save it for later use.
- Create a SageMaker Batch Transform job to get inferences for a large dataset.
- Automate the entire batch inference process using EventBridge and Lambda.
- Implement a solution that is serverless and cost-optimized, as resources are only provisioned when the job runs.
- Use AWS Glue and Athena to query and validate the inference results.

---

2. Architecture

The architecture is an event-driven, serverless workflow designed for efficient, large-scale data processing.



Workflow Steps:

1. **Data Arrival:** A large dataset (e.g., a CSV file with millions of records) is uploaded to a specific `input` prefix in an S3 bucket.
2. **Pipeline Trigger:** An EventBridge rule is configured to detect the `Object Created` event in the S3 input path. This rule triggers a Lambda function.
3. **Start Batch Job:** The Lambda function initiates a SageMaker Batch Transform job. It passes the S3 path of the newly arrived data as input and specifies an S3 path for the output.
4. **Batch Inference:** SageMaker automatically provisions the necessary compute resources (e.g., `ml.m5.large` instances), loads the pre-trained model, runs inference on the entire input dataset, and then terminates the resources. This is highly cost-effective.
5. **Store Results:** The Batch Transform job writes the prediction results to the specified `output` prefix in the S3 bucket.
6. **Error Handling:** If the job fails, the error logs are directed to a separate `error` path in S3, and an SNS topic can be triggered to alert the operations team.
7. **Result Analysis:** An AWS Glue Crawler can be run on the output data to catalog it. Amazon Athena can then be used to query the results for validation, reporting, or further analysis.

---

3. Prerequisites

- An AWS account with permissions for SageMaker, S3, IAM, Lambda, EventBridge, Glue, and Athena.
- The AWS CLI installed and configured.
- A SageMaker Studio domain.

---

4. Step-by-Step Deployment Guide

This guide is designed to be run within a **SageMaker Studio Notebook**.

Step 4.1: Train and Save a Model

First, we need a model to use for the batch transform job. We will train a simple linear regression model on the Abalone dataset.

```
# Cell 1: Setup and Data Prep
import sagemaker
import boto3
import pandas as pd
from sklearn.model_selection import train_test_split

sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = sess.boto_region_name

prefix = "abalone-batch-inference"

# Download data
!wget https://archive.ics.uci.edu/ml/machine-learning-
databases/abalone/abalone.data

# Prepare data
column_names = ["sex", "length", "diameter", "height", "whole_weight",
"shucked_weight", "viscera_weight", "shell_weight", "rings"]
data = pd.read_csv("abalone.data", names=column_names)
data["sex"] = data["sex"].astype("category").cat.codes # Convert categorical
to numerical

# Split data for training and for batch inference
train, batch_data = train_test_split(data, test_size=0.3, random_state=42)

# Save and upload
train.to_csv("train.csv", index=False, header=False)
batch_data.to_csv("batch_input.csv", index=False, header=False)

s3_train_path = sess.upload_data("train.csv", bucket=bucket, key_prefix=f"
{prefix}/train")
s3_batch_input_path = sess.upload_data("batch_input.csv", bucket=bucket,
key_prefix=f"{prefix}/input")

print(f"Training data: {s3_train_path}")
print(f"Batch input data: {s3_batch_input_path}")

# Cell 2: Train a Linear Learner Model
from sagemaker.image_uris import retrieve

container = retrieve("linear-learner", region)

ll_estimator = sagemaker.estimator.Estimator(container,
```

```
                                            role,
                                            instance_count=1,
                                            instance_type="ml.m5.large",

output_path=f"s3://{bucket}/{prefix}/output",
                                            sagemaker_session=sess)

ll_estimator.set_hyperparameters(predictor_type="regressor")

# We need to format the data as RecordIO-Protobuf
train_records = sagemaker.inputs.TrainingInput(s3_train_path,
content_type="text/csv")

ll_estimator.fit({"train": train_records})

# Cell 3: Create a SageMaker Model from the trained estimator
model_name = f"{prefix}-model-{int(time.time())}"
model = ll_estimator.create_model(name=model_name)

print(f"Model '{model.name}' created.")
```

Step 4.2: Create the Batch Transform Job Manually (for testing)

Before automating, let's run one job manually to ensure the model works as expected.

```
# Cell 4: Create a Transformer object
transformer = ll_estimator.transformer(
    instance_count=1,
    instance_type="ml.m5.large",
    output_path=f"s3://{bucket}/{prefix}/batch-output",
    accept="text/csv",
    strategy="SingleRecord"
)

# Start the transform job
transformer.transform(s3_batch_input_path, content_type="text/csv",
split_type="Line")

transformer.wait()

print(f"Batch transform job complete. Output is in
{transformer.output_path}")
```

After this completes, you can check the S3 output path to see the prediction results.

Step 4.3: Create the Automation Components (Lambda and EventBridge)

1. **Create IAM Role for Lambda:**
    - Create a role named `BatchInferenceLambdaRole`.
    - **Permissions:** `AWSLambdaBasicExecutionRole`, `AmazonSageMakerFullAccess`, `AmazonS3FullAccess`.
2. **Create the Lambda Function:**
    - **Name:** `start-batch-transform-job`
    - **Runtime:** Python 3.9
    - **Role:** `BatchInferenceLambdaRole`
    - **Code:**

```
import boto3
import os
import time


sm_client = boto3.client("sagemaker")


def lambda_handler(event, context):
    # Get bucket and key from the S3 event
    s3_bucket = event["detail"]["bucket"]["name"]
    s3_key = event["detail"]["object"]["key"]
    input_path = f"s3://{s3_bucket}/{s3_key}"


    # Define job parameters
    job_name = f"batch-job-{int(time.time())}"
    model_name = os.environ["MODEL_NAME"]
    output_path = f"s3://{s3_bucket}/batch-inference/output/"


    print(f"Starting batch transform job {job_name} for input
{input_path}")


    sm_client.create_transform_job(
        TransformJobName=job_name,
        ModelName=model_name,
        TransformInput={
            "DataSource": {"S3DataSource": {"S3DataType":
"S3Prefix", "S3Uri": input_path}},
            "ContentType": "text/csv",
            "SplitType": "Line"
        },
        TransformOutput={"S3OutputPath": output_path, "Accept":
"text/csv"},
        TransformResources={"InstanceType": "ml.m5.large",
"InstanceCount": 1}
    )


    return {"statusCode": 200, "body": f"Started job {job_name}"}
```

- **Environment Variables:**

- `MODEL_NAME`: The name of the SageMaker model you created in Cell 3 (e.g., `abalone-batch-inference-model-<timestamp>`).

3. **Create the EventBridge Rule:**
   - Go to the **Amazon EventBridge Console** -> **Create rule**.
   - **Name:** `trigger-batch-inference-on-s3-upload`
   - **Event pattern:**

```
{
  "source": ["aws.s3"],
  "detail-type": ["Object Created"],
  "detail": {
    "bucket": {
      "name": ["<YOUR_BUCKET_NAME>"]
    },
    "object": {
      "key": [{
        "prefix": "batch-inference/input/"
      }]
    }
  }
}
```

     Important: Replace `<YOUR_BUCKET_NAME>` with your actual SageMaker default bucket name.
   - **Target:** Select the `start-batch-transform-job` Lambda function.
   - Create the rule.

---

5. How to Use the Automated Pipeline

1. **Upload a new data file** to the specific S3 prefix that EventBridge is monitoring.

```
# Create another dummy data file
echo "2,0.5,0.4,0.1,0.6,0.2,0.1,0.2" > new_batch_data.csv

# Upload it to the correct path
aws s3 cp new_batch_data.csv s3://<YOUR_BUCKET_NAME>/batch-
inference/input/new_batch_data.csv
```

Note: Replace `<YOUR_BUCKET_NAME>` with your bucket name.

2. **Monitor the process:**
    - The upload will trigger the EventBridge rule.
    - The rule will invoke the Lambda function.
    - The Lambda function will start a new SageMaker Batch Transform job.
    - You can see the new job running in the **SageMaker Console -> Batch Transform Jobs**.
    - Once complete, the output will appear in the `s3://<YOUR_BUCKET_NAME>/batch-inference/output/` folder.

Step 5.1: Analyze Results with Athena

1. **Create a Glue Crawler:**
    - **Name:** `batch-output-crawler`
    - **Data source:** `s3://<YOUR_BUCKET_NAME>/batch-inference/output/`
    - **IAM Role:** Create a new role for Glue.
    - **Database:** Create a new database `sagemaker_batch_db`.
    - Run the crawler.
2. **Query with Athena:**
    - Go to the Athena console.
    - Select the `sagemaker_batch_db` database.
    - Query the newly created table:

```
SELECT * FROM "output" LIMIT 100;
```

This allows you to easily analyze the millions of predictions generated by your batch job.

---

6. Cleanup

1. **Delete the EventBridge rule and Lambda function.**
2. **Delete the Glue crawler and database.**
3. **Delete the SageMaker model:**

```
# In your SageMaker notebook
sm_client = boto3.client("sagemaker")
sm_client.delete_model(ModelName=model.name)
```

4. **Empty and delete the S3 bucket** to remove all data, outputs, and artifacts.

```
aws s3 rb s3://<your-bucket-name> --force
```

Business Context

The Problem

Organizations struggle to deploy ML models securely and at scale. ML workflows lack security controls, model training data is not protected, and model endpoints are vulnerable to attacks. Manual ML operations are slow and error-prone, preventing rapid iteration and deployment.

The Solution

Secure MLOps pipeline with automated model training, validation, and deployment. Implements data encryption, model versioning, endpoint security, and monitoring. Provides reproducible ML workflows with security and compliance built-in from the start.

Business Value

- **Accelerated ML Deployment:** Reduces model deployment time from weeks to days
- **Data Protection:** Encrypts training data and model artifacts at rest and in transit
- **Model Governance:** Complete audit trail of model versions, training data, and performance
- **Scalable Infrastructure:** Auto-scaling endpoints handle production traffic efficiently

Risk Mitigation

Protects sensitive training data, prevents model theft, ensures model integrity, and maintains compliance with data privacy regulations.

GRC Mapping

Compliance Frameworks

- **NIST AI RMF:** Govern, Map, Measure, Manage functions
- **ISO 27001:** A.9.4 (System access control), A.14.2 (Security in development)
- **NIST CSF:** PR.DS-1 (Data protection), ID.RA-1 (Asset vulnerabilities)
- **AWS Well-Architected ML Lens:** Security and operational excellence

Security Controls Implemented

- Data encryption at rest and in transit
- Model versioning and artifact management

- Endpoint authentication and authorization
- Model monitoring and drift detection
- Training data access controls

Audit Evidence

- Model training logs and hyperparameters
- Data lineage and provenance records
- Model performance metrics and validation results
- Endpoint access logs and API calls

Regulatory Alignment

- **GDPR:** Article 22 (Automated decision-making), Article 35 (Data protection impact assessment)
- **HIPAA:** § 164.308(a)(3) (Workforce access), § 164.312(a)(1) (Access control)
- **AI Act (EU):** High-risk AI system requirements
- **SOC 2:** CC6.1 (Logical access), CC7.2 (System monitoring)