

Telco Churn: Lightweight Inference Pipeline

1. Executive Summary

I successfully deployed the churn model as an automated AWS pipeline connected to my original code. Using AWS S3 and Lambda, I built a serverless inference system that loads the lightweight model artifact from main code and returns predictions via API Gateway.

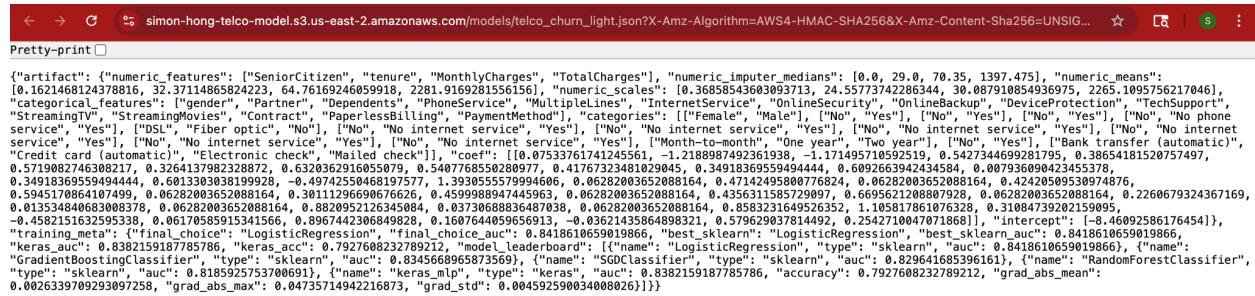
2. Approach & Methodology

I uploaded the lightweight artifact to AWS S3, used AWS Lambda with boto3 library to load the artifact from S3, parse the input payload, and return predictions via API Gateway. This architecture aims to extend this project beyond local analysis, but implement a serverless, automated inference pipeline. This system allows real-time churn predictions on new customer data without requiring persistent server or local dependencies. The aim was lightweight, where we used an extracted linear model.

Architecture

The architecture follows the pipeline flow of S3, Lambda, and API Gateway. S3 serves as the model registry where the trained model parameters including weights, intercepts, and scaling factors are transformed to a JSON artifact and uploaded to S3 bucket. Then, the function in Lambda is triggered, which downloads the artifact and reconstructs the model to compute probability. Then, the API Gateway takes the Lambda function to the public and allows the client to request prediction.

Figure 1: Lightweight JSON artifact uploaded to S3



Implementation

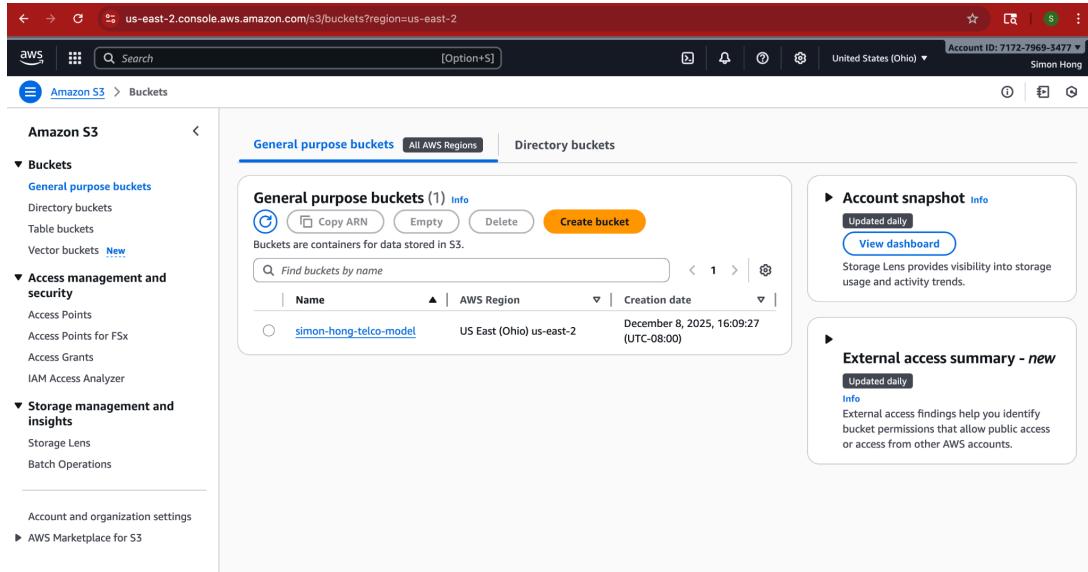
The key to this challenge was to keep everything lightweight for easy deployment. That means it is difficult to install libraries to AWS Lambda. So, I trained an SGD classifier locally using

Scikit-Learn, extracted the key coefficients and statistics through the artifact, and let Lambda do this directly.

Amazon S3

I created a bucket with the name ‘simon-hong-telco-model’. This works as the centralized model registry, which automatically uploads the artifact when the main script from my local device finishes.

Figure 2: Amazon S3 bucket



AWS Lambda

I set the runtime to Python 3.14. Custom lambda functions including `lambda_handler` is used to fetch the JSON artifact from S3, which then applies the linear formula for SGD Classifier, and returns the prediction. I granted IAM permission to allow reading access to S3, and connect API Gateway as a method to stream the result out.

Figure 3: AWS Lambda

The screenshot shows the AWS Lambda console interface. At the top, the URL is `us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions/telco-churn-lambda?subtab=triggers&tab=code`. The top bar includes the AWS logo, search bar, and account information (Account ID: 7172-7969-3477, Simon Hong). The main navigation bar shows 'Lambda > Functions > telco-churn-lambda'. The left sidebar has sections for 'Function overview' (selected), 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' section is currently active. The main content area shows the function details: 'telco-churn-lambda' (ARN: arn:aws:lambda:us-east-2:717279693477:function:telco-churn-lambda), last modified 1 hour ago, and a description. It also shows an API Gateway trigger and a 'Layers' section (0 layers). On the right, there is a 'Tutorials' sidebar titled 'Create a simple web app' with a link to learn more and start the tutorial.

Figure 4: AWS Lambda IAM Permissions

The screenshot shows the AWS IAM console interface. The URL is `us-east-1.console.aws.amazon.com/iam/home#/roles/details/telco-churn-lambda-role-hm3yvq4r`. The top bar includes the AWS logo, search bar, and account information (Account ID: 7172-7969-3477, Simon Hong). The left sidebar has sections for 'Identity and Access Management (IAM)', 'Access management', and 'Access reports'. The 'Access management' section is active, showing 'Roles' (selected) and 'telco-churn-lambda-role-hm3yvq4r'. The main content area shows a message 'Policy removed.' with details: Creation date December 09, 2025, 16:16 (UTC-08:00), ARN arn:aws:iam:717279693477:role/service-role/telco-churn-lambda-role-hm3yvq4r, Last activity 17 minutes ago, and Maximum session duration 1 hour. Below this, the 'Permissions' tab is selected, showing a table of 'Permissions policies (2)'. The table includes columns for Policy name, Type, and Attached entities. Policies listed: 'AmazonS3ReadOnlyAccess' (AWS managed) and 'AWSLambdaBasicExecutionRole-592d0...' (Customer managed).

Figure 5: Custom lambda function uploaded to AWS Lambda

```

    import os
    import json
    import math
    import boto3
    s3 = boto3.client("s3")
    _ARTIFACT = None
    _WEIGHTS = None
    _INTERCEPT = None
    global _ARTIFACT, _WEIGHTS, _INTERCEPT
    if _ARTIFACT is not None:
        return _ARTIFACT
    obj = s3.get_object(Bucket=MODEL_BUCKET, Key=MODEL_KEY)
    body = obj["Body"].read().decode("utf-8")
    wrapped_artifact = json.loads(body)
    artifact = wrapped_artifact["artifact"]
    weights = artifact["com#"][@]
    intercept = artifact["intercept"][@]
    _ARTIFACT = artifact
    _WEIGHTS = weights
    _INTERCEPT = intercept

```

API Gateway

I used HTTP API, which provides a public URL to trigger the Lambda function securely over AWS.

Workflow

The pipeline is fully automated within the lambda functions in the main script `telco_churn_final.py`. The workflow starts at training and building the SGD Classifier from the main script. Then, the `upload_json_to_s3()` function builds and uploads it to S3 using the `boto3` library. The script then automatically selects a random sample customer for testing. The script sends POST requests to API Gateway URL with the customer data, where the Lambda processes the request using uploaded artifact and returns the Churn probability and label.

Figure 6: Automated AWS Pipeline execution result

```
✓ TERMINAL          zsh + ~
```

```
FINAL CHOICE (for your report): GradientBoostingClassifier AUC=0.8434
Saved model_leaderboard_telco.json

ENABLE_AWS_EXPORT=True → building lightweight artifact
and uploading to S3.
Uploaded to s3://simon-hong-telco-model/models/telco_churn_light.json
Uploaded lightweight artifact with metadata to S3.
SUCCESS. The API returned:
{'predictions': [0.005088353071451572], 'labels': [0]}
```

This figure serves as the verification of success for the end-to-end pipeline. The local training script automatically triggered the API with a sample customer, and returned a prediction with the correct label, confirming infrastructure is fully operational.