Bring Your Own Model with SageMaker Batch Transform

Table of Contents

1 Problem Statement

Deploying a custom ML model using SageMaker particularly for the customers who begin their cloud journey with AWS or had never used SageMaker before, can be a long learning curve taking extensive effort to adopt. It's easier for customers to use SageMaker built-in algorithms but a key concern that many customers have if they want to continue using their existing pre-trained model. The next challenge of deploying custom models is making sure that they are able to scale and meet increases in performance and application demand in production, particularly if dataset is large or need beefy GPUs for image processing then they need more computing power for generating inferences. Many customers abandon their proposal of doing a PoC for moving from on-prem to AWS before they ever make it into production because of complexities of deployment at scale that slow down or halt the entire process.

2 Solution Overview

This solution allows you to get inferences from large datasets with bring your own Model using **AWS Batch** and **SageMaker with batch transform**. In this example we are using a trained XGBoost model with supervised learning to build a credit card fraud detection system. However, this solution can be modified to support other types of trained models with different type of algorithms like deep learning object detection models.

It deploys the entire solution with CloudFormation using a bash script.

For getting the inference from the SageMaker and possible batch processing transformations it builds an Amazon ECR image with a Python-based execution script and support libraries into an image and executes on AWS Batch.

When an input data is uploaded to S3 bucket, it triggers a series of events to get the inference. SageMaker saves the inferences in the output directory of the S3 bucket.

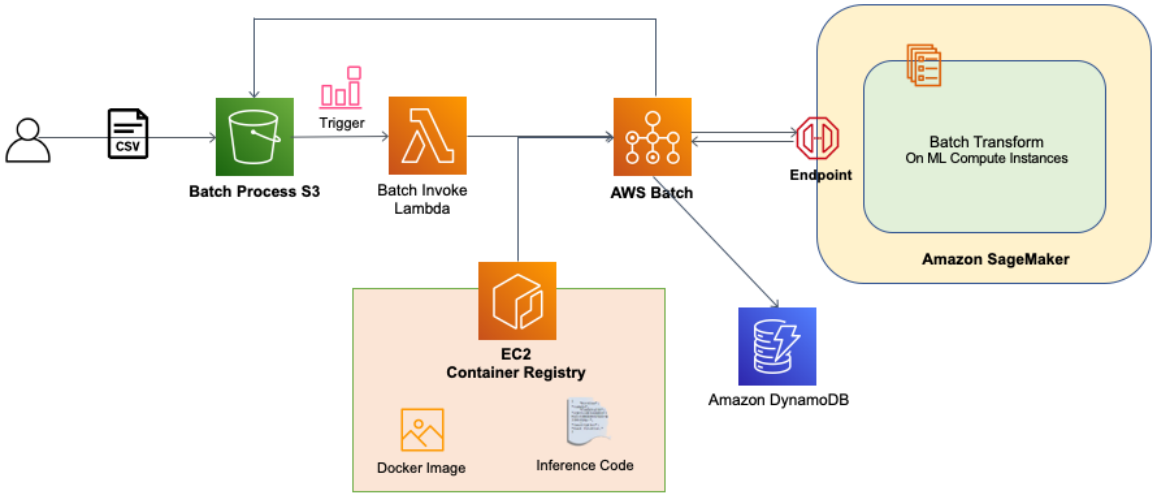
3 Batch Transform Architecture

3.1 Why AWS Batch with SageMaker?

For a Batch Transform job, before or after invoking the SageMaker endpoint it might be necessary to use compute resources to transform the input or output data on the fly based on the type of inference e.g adding a bounding box with the inference results to an output image, extracting frames or images from an input video file, adding metadata to output CSV file etc. It's important to optimize the job distribution based on the volume and resource requirements of the batch processing. AWS Batch provisions compute resources dynamically and scales compute resources to any quantity with the ability to configure such as min/max/desired vCPUs. It also can provide cost-effective batch processing by taking advantage of SPOT instances.

3.2 Batch Processing Workflow

The following diagram shows the workflow for the inference.



- User uploads the input csv file to the input folder of the S3 bucket. It contains the data to send requests to the model for inferences.
- The file upload triggers a lambda function that invokes the batch job.
- The batch job spins up the compute resources using the docker image and inference code from the EC2 container registry.
- The compute resource of the batch job then executes the inference code which in turn invokes the SageMaker endpoint, gets the inference, transform the results and uploads the output file to the same S3 bucket in the output folder.
- Writes the status of the process job in Dynamo DB

4 Deployment using CloudFormation

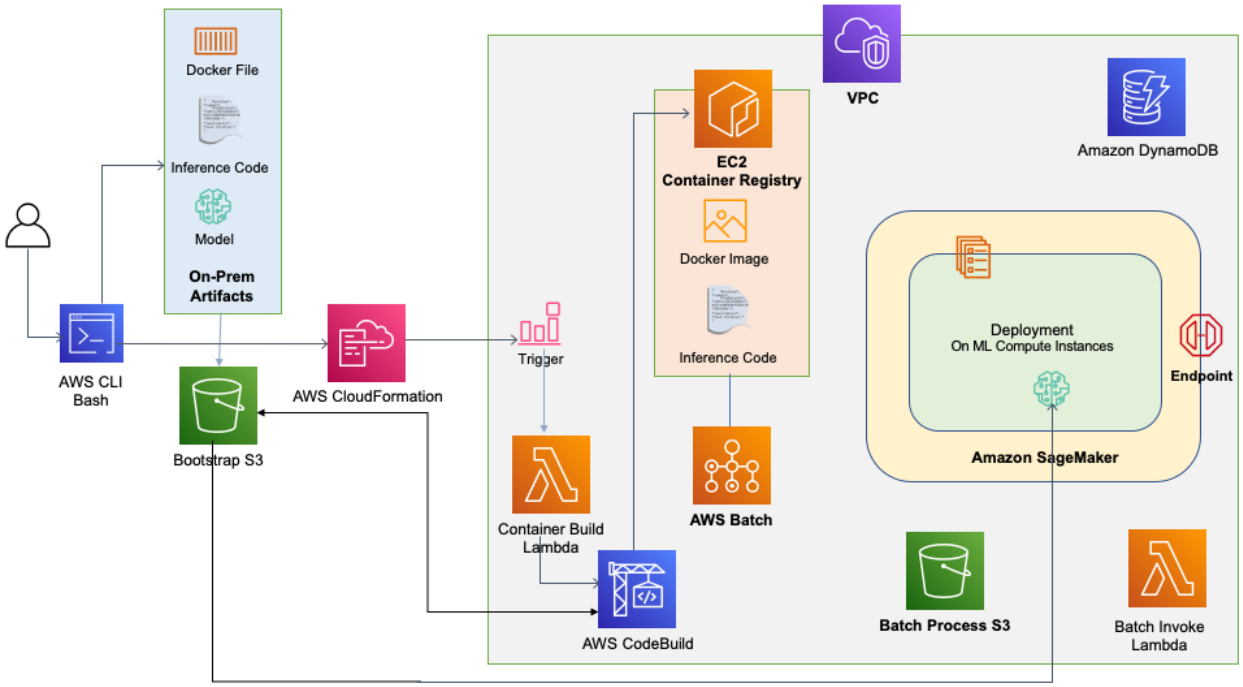
4.1 The project artifacts

- **BYO model**: Trained model with supervised learning
- **Container**: Container definition that will host the code to invoke the SageMaker endpoint.
- **CloudFormation**: CloudFormation template needed to deploy the infrastructure.
- **Dataset**: This contains original labeled dataset which was used for training. It also contains a sample dataset to validate the model after deployment.
  The original dataset contains only numerical features, because the original features have been transformed for confidentiality using PCA. The dataset used to demonstrate the fraud detection solution has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data

mining and fraud detection. https://github.com/awslabs/fraud-detection-using-machine-learning/blob/master/source/notebooks/sagemaker_fraud_detection.ipynb

4.2 Deployment workflow

The following diagram shows the workflow of the deployment steps:



- The on-prem artifacts – trained model and docker image with python file (code to get the predictions) can be copied to the local working directory.
- The shell script packages create an interim bootstrap S3 bucket and uploads the artifacts to the bucket.
- The shell script then deploys the CloudFormation stack.
- The CloudFormation builds the following key resources in a new VPC:
    - SageMaker endpoint.
    - S3 bucket for where you will store the data that you want to predict.
    - CodeBuild project to build and push the Docker image
    - Lambda Function to trigger the CodeBuild
    - AWS Batch to get the inference and transform. It will launch the docker container to execute the python code that would include the API for invoking the SageMaker endpoint.
    - Lambda Function that will be used to trigger the Batch Job when a data is uploaded
- After the ClodFormation stack is created it will invoke a lambda function which will trigger the CodeBuild.
- The CodeBuild will build and push the Docker image to the EC2 Container Registry

4.3 Code Repo

The code can be downloaded from the link as below. It contains a README.md for the details.

https://amazon.awsapps.com/workdocs/index.html#/folder/cddb9df9d6108c6e3afd4f83713f45538cc9235a91f48f811e2a86cec52a2390

5 Validate the Deployed Model

5.1 Getting Started

Once the deployment is completed a user can upload the input data to S3 bucket. The inferences are then automatically generated and saved in the output directory of the same S3 bucket.

The solution contains labeled dataset including sample dataset to validate the model. It contains only numerical features, because the original features have been transformed for confidentiality using PCA.

5.2 Steps to perform

a. Go to S3 bucket batch-processing-job-<Region>-<Account ID>
b. Create a folder with name as "input"
c. Upload the sample CSV from ml-byo/sample/sample.csv to input folder of the S3 bucket. This will trigger the batch job which will invoke the SageMaker endpoint. The output file will be saved in the S3 bucket.
d. After few minutes you should see the output folder in the S3 bucket. Navigate to the output folder and download the <timestamp>-sample.csv file.
e. The output should look like as indicated below, with each record corresponds to each row of the input file.
0.000260044, 0.000412545, 0.001004696, 0.0000873892495292, 0.977230012, 0.970581293, 0.445608526
In this example each record is classified as normal (class "0") or fraudulent (class "1")

6 Acknowledgements

- Similar architecture has been designed for ongoing AWS Olympics for Intel using 3DAT modeling.
- The dataset used to demonstrated the fraud detection solution has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection
- The custom model was trained using the following notebook https://github.com/awslabs/fraud-detection-using-machine-learning/blob/master/source/notebooks/sagemaker_fraud_detection.ipynb