Safe MLOps Deployment Pipeline

Overview

In this notebook you will step through an MLOps pipeline to build, train, deploy and monitor an XGBoost regression model for predicting the expected taxi fare using the New York City Taxi dataset (https://registry.opendata.aws/nyc-tlc-trip-records-pds/)⇗. This safe pipeline features a canary deployment (https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/canary-deployment.html)⇗ strategy with rollback on error. You will learn how to trigger and monitor the pipeline, inspect the training workflow, use model monitor to set up alerts, and create a canary deployment.

Note: This notebook assumes prior familiarity with the basics training ML models on Amazon SageMaker. Data preparation and visualization, although present, will be kept to a minimum. If you are not familiar with the basic concepts and features of SageMaker, we recommend reading the SageMaker documentation (https://docs.aws.amazon.com/sagemaker/)⇗ and completing the workshops and samples in AWS SageMaker Examples GitHub (https://github.com/aws/amazon-sagemaker-examples)⇗ and AWS Samples GitHub (https://github.com/aws-samples?q=sagemaker&type=&language=)⇗.

Contents

This notebook has the following key sections:

Architecture

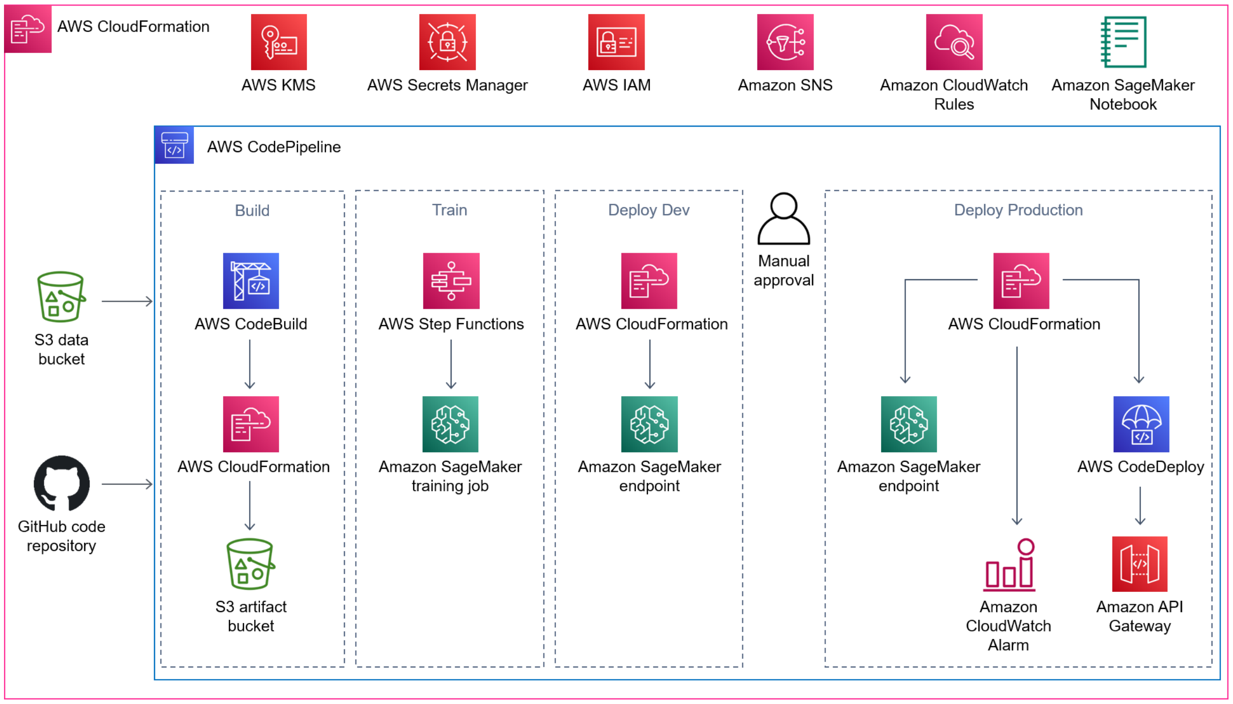
The architecture diagram below shows the entire MLOps pipeline at a high level.

Use the CloudFormation template provided in this repository (`pipeline.yml`) to build the demo in your own AWS account. If you are currently viewing this notebook from SageMaker in your AWS account, then you have already completed this step. CloudFormation deploys several resources:

1. A customer-managed encryption key in in Amazon KMS for encrypting data and artifacts.
2. A secret in Amazon Secrets Manager to securely store your GitHub Access Token.
3. Several AWS IAM roles so CloudFormation, SageMaker, and other AWS services can perform actions in your AWS account, following the principle of least privilege (https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege)⇗.

4. A messaging service in Amazon SNS to notify you when CodeDeploy has successfully deployed the API, and to receive alerts for retraining and drift detection (signing up for these notifications is optional).
5. Two Amazon CloudWatch event rules: one which schedules the pipeline to run every month, and one which triggers the pipeline to run when SageMaker Model Monitor detects certain metrics.
6. An Amazon SageMaker Jupyter notebook with this workshop content pre-loaded.
7. An Amazon S3 bucket for storing model artifacts.
8. An AWS CodePipeline instance with several pre-defined stages.

Take a moment to look at all of these resources now deployed in your account.



In this notebook, you will work through the CodePipeline instance created by the CloudFormation template. It has several stages:

1. **Source** - The pipeline is already configured with two sources. If you upload a new dataset to a specific location in the S3 data bucket, this will trigger the pipeline to run. The Git source can be GitHub, or CodeCommit if you don't supply your access token. If you commit new code to your repository, this will trigger the pipeline to run.
2. **Build** - In this stage, CodeBuild configured by the build specification `model/buildspec.yml` will execute `model/run.py` to generate AWS CloudFormation templates for creating the AWS Step Function (including AWS Lambda custom resources), and deployment templates used in the following stages based on the data sets and hyperparameters specified for this pipeline run. You will take a closer look at these files later in this notebook.
3. **Train** The Step Functions workflow created in the Build stage is run in this stage. The workflow creates a baseline for the model monitor using a SageMaker processing job, and trains an XGBoost model on the taxi ride dataset using a SageMaker training job.
4. **Deploy Dev** In this stage, a CloudFormation template created in the build stage (from `assets/deploy-model-dev.yml`) deploys a dev endpoint. This will allow you to run tests on the model and decide if the model is of sufficient quality to deploy into production.

5. **Deploy Production** The final stage of the pipeline is the only stage which does not run automatically as soon as the previous stage is complete. It waits for a user to manually approve the model which was previously deployed to dev. As soon as the model is approved, a CloudFormation template (packaged from `assets/deploy-model-prod.yml` to include the Lambda functions saved and uploaded as ZIP files in S3) deploys the production endpoint. It configures autoscaling and enables data capture. It creates a model monitoring schedule and sets CloudWatch alarms for certain metrics. It also sets up an AWS CodeDeploy instance which deploys a set of AWS Lambda functions and an Amazon API Gateway to sit in front of the SageMaker endpoint. This stage can make use of canary deployment to safely switch from an old model to a new model.

In [ ]:
```
# Import the latest sagemaker and boto3 SDKs.
import sys
!{sys.executable} -m pip install --upgrade pip
!{sys.executable} -m pip install -qU awscli boto3 "sagemaker>=2.1.0<3" tqdm
!{sys.executable} -m pip install -qU "stepfunctions==2.0.0"
!{sys.executable} -m pip show sagemaker stepfunctions
```

Restart your SageMaker kernel then continue with this notebook.

Data Prep

In this section of the notebook, you will download the publicly available New York Taxi dataset in preparation for uploading it to S3.

Download Dataset

First, download a sample of the New York City Taxi dataset (https://registry.opendata.aws/nyc-tlc-trip-records-pds/)⇗ to this notebook instance. This dataset contains information on trips taken by taxis and for-hire vehicles in New York City, including pick-up and drop-off times and locations, fares, distance traveled, and more.

In [1]:
```
!aws s3 cp 's3://mlops-nyctaxi-artifact-us-east-1-929911052415/green_tripdata_20
```

```
download: s3://mlops-nyctaxi-artifact-us-east-1-929911052415/green_tripdata_201
8-02.csv to ./nyc-tlc.csv
```

Now load the dataset into a pandas data frame, taking care to parse the dates correctly.

In [2]:
```
import pandas as pd

parse_dates= ['lpep_dropoff_datetime', 'lpep_pickup_datetime']
trip_df = pd.read_csv('nyc-tlc.csv', parse_dates=parse_dates)

trip_df.head()
```

Out[2]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULoca
0	2	2018-02-01 00:39:38	2018-02-01 00:39:41	N	5	
1	2	2018-02-01 00:58:28	2018-02-01 01:05:35	N	1	
2	2	2018-02-01 00:56:05	2018-02-01 01:18:54	N	1	
3	2	2018-02-01 00:12:40	2018-02-01 00:15:50	N	1	
4	2	2018-02-01 00:45:18	2018-02-01 00:51:56	N	1	

Data manipulation

Instead of the raw date and time features for pick-up and drop-off, let's use these features to calculate the total time of the trip in minutes, which will be easier to work with for our model.

In [3]:
```
trip_df['duration_minutes'] = (trip_df['lpep_dropoff_datetime'] - trip_df['lpep_
```

The dataset contains a lot of columns we don't need, so let's select a sample of columns for our machine learning model. Keep only `total_amount` (fare), `duration_minutes` , `passenger_count` , and `trip_distance` .

In [4]:
```
cols = ['total_amount', 'duration_minutes', 'passenger_count', 'trip_distance']
data_df = trip_df[cols]
print(data_df.shape)
data_df.head()
```

```
(769940, 4)
```

Out[4]:

	total_amount	duration_minutes	passenger_count	trip_distance
0	23.00	0.050000	1	0.00
1	9.68	7.116667	5	1.60
2	35.76	22.816667	1	9.60
3	5.80	3.166667	1	0.73
4	9.30	6.633333	2	1.87

Generate some quick statistics for the dataset to understand the quality.

In [5]:
```
data_df.describe()
```

Out[5]:

	total_amount	duration_minutes	passenger_count	trip_distance
count	769940.000000	769940.000000	769940.000000	769940.000000
mean	14.156694	19.536958	1.356391	2.725387
std	10.707214	93.847137	1.033778	2.882936
min	-400.000000	0.000000	0.000000	0.000000
25%	7.800000	6.016667	1.000000	1.000000
50%	11.150000	10.116667	1.000000	1.750000
75%	17.000000	16.850000	1.000000	3.370000
max	2626.300000	1439.883333	9.000000	120.470000

The table above shows some clear outliers, e.g. -400 or 2626 as fare, or 0 passengers. There are many intelligent methods for identifying and removing outliers, but data cleaning is not the focus of this notebook, so just remove the outliers by setting some min and max values which seem more reasonable. Removing the outliers results in a final dataset of 754,671 rows.

In [6]:
```
data_df = data_df[(data_df.total_amount > 0) & (data_df.total_amount < 200) &
                  (data_df.duration_minutes > 0) & (data_df.duration_minutes < 1
                  (data_df.trip_distance > 0) & (data_df.trip_distance < 121) &
                  (data_df.passenger_count > 0)].dropna()
print(data_df.shape)
```

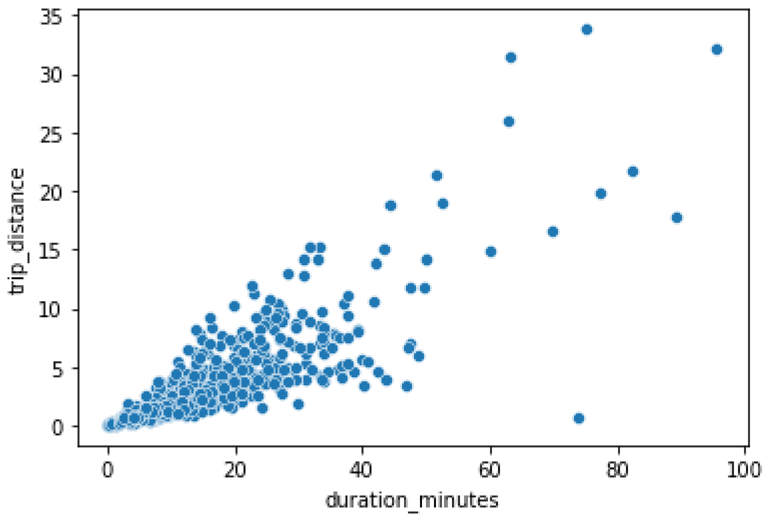
```
(754671, 4)
```

Data visualization

Since this notebook will build a regression model for the taxi data, it's a good idea to check if there is any correlation between the variables in our data. Use scatter plots on a sample of the data to compare trip distance with duration in minutes, and total amount (fare) with duration in minutes.

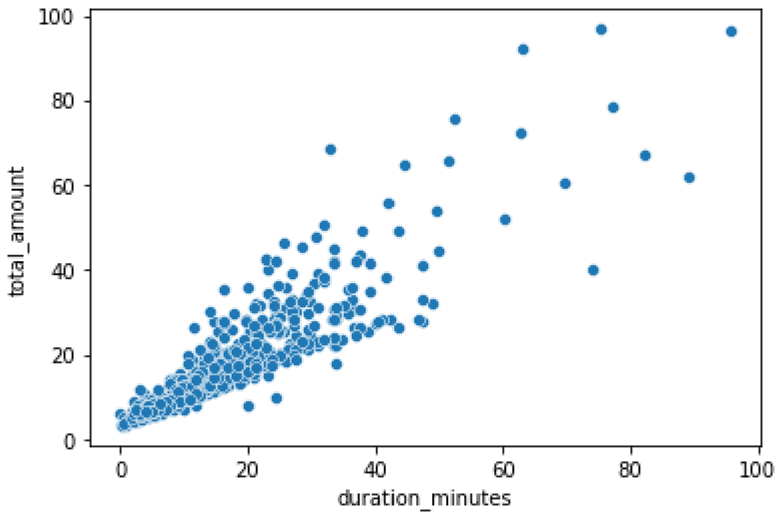
In [7]:
```
import seaborn as sns

sample_df = data_df.sample(1000)
sns.scatterplot(data=sample_df, x='duration_minutes', y='trip_distance')
```

Out[7]:
```
<AxesSubplot:xlabel='duration_minutes', ylabel='trip_distance'>
```



In [8]:
```
sns.scatterplot(data=sample_df, x='duration_minutes', y='total_amount')
```

Out[8]:
```
<AxesSubplot:xlabel='duration_minutes', ylabel='total_amount'>
```



These scatter plots look fine and show at least some correlation between our variables.

Data splitting and saving

We are now ready to split the dataset into train, validation, and test sets.

In [9]:
```
from sklearn.model_selection import train_test_split
train_df, val_df = train_test_split(data_df, test_size=0.20, random_state=42)
val_df, test_df = train_test_split(val_df, test_size=0.05, random_state=42)

# Reset the index for our test dataframe
test_df.reset_index(inplace=True, drop=True)

print('Size of\n train: {},\n val: {},\n test: {} '.format(train_df.shape[0], va
```

```
Size of
 train: 603736,
 val: 143388,
 test: 7547
```

Save the train, validation, and test files as CSV locally on this notebook instance. Notice that you save the train file twice - once as the training data file and once as the baseline data file. The baseline data file will be used by SageMaker Model Monitor (https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html) to detect data drift. Data drift occurs when the statistical nature of the data that your model receives while in production drifts away from the nature of the baseline data it was trained on, which means the model begins to lose accuracy in its predictions.

In [10]:
```
train_cols = ['total_amount', 'duration_minutes','passenger_count','trip_distanc
train_df.to_csv('train.csv', index=False, header=False)
val_df.to_csv('validation.csv', index=False, header=False)
test_df.to_csv('test.csv', index=False, header=False)

# Save test and baseline with headers
train_df.to_csv('baseline.csv', index=False, header=True)
```

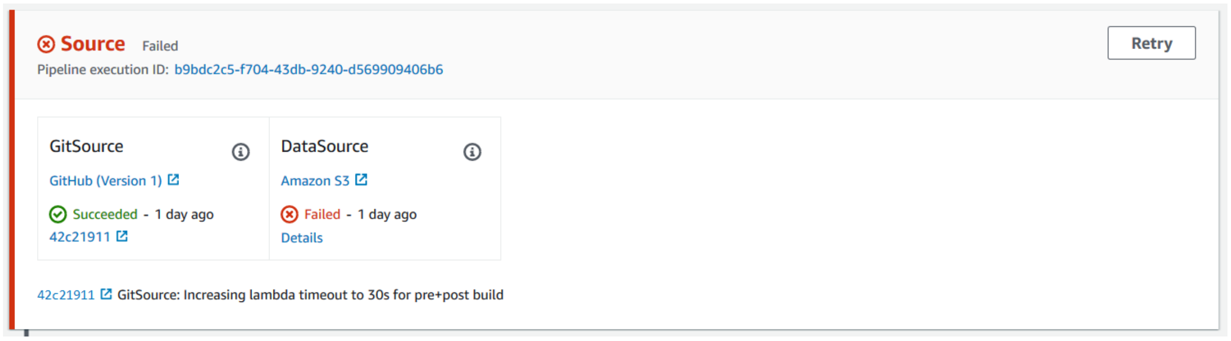
Now upload these CSV files to your default SageMaker S3 bucket.

In [11]:
```
import sagemaker

# Get the session and default bucket
session = sagemaker.session.Session()
bucket = session.default_bucket()

# Specify data prefix and version
prefix = 'nyc-tlc/v1'

s3_train_uri = session.upload_data('train.csv', bucket, prefix + '/data/training
s3_val_uri = session.upload_data('validation.csv', bucket, prefix + '/data/valid
s3_test_uri = session.upload_data('test.csv', bucket, prefix + '/data/test')
s3_baseline_uri = session.upload_data('baseline.csv', bucket, prefix + '/data/ba
```

You will use the datasets which you have prepared and saved in this section to trigger the pipeline to train and deploy a model in the next section.

Build

If you navigate to the CodePipeline instance created for this workshop, you will notice that the Source stage is initially in a `Failed` state. This happens because the dataset, which is one of the sources that can trigger the pipeline, has not yet been uploaded to the S3 location expected by the pipeline.



Pipeline execution ID: b9bdc2c5-f704-43db-9240-d569909406b6

GitSource: Succeeded - 1 day ago, 42c21911

DataSource: Failed - 1 day ago

42c21911 GitSource: Increasing lambda timeout to 30s for pre+post build

Trigger Build

In this section, you will start a model build and deployment pipeline by packaging up the datasets you prepared in the previous section and uploading these to the S3 source location which triggers the CodePipeline instance created for this workshop.

First, import some libraries and load some environment variables which you will need. These environment variables have been set through a lifecycle configuration (https://docs.aws.amazon.com/sagemaker/latest/dg/notebook-lifecycle-config.html)↗ script attached to this notebook.

In [52]:
```python
import boto3
from botocore.exceptions import ClientError
import os
import time

region = boto3.Session().region_name
artifact_bucket = os.environ['ARTIFACT_BUCKET']
pipeline_name = os.environ['PIPELINE_NAME']
model_name = os.environ['MODEL_NAME']
workflow_pipeline_arn = os.environ['WORKFLOW_PIPELINE_ARN']

print('region: {}'.format(region))
print('artifact bucket: {}'.format(artifact_bucket))
print('pipeline: {}'.format(pipeline_name))
print('model name: {}'.format(model_name))
print('workflow: {}'.format(workflow_pipeline_arn))
```

```
region: us-east-1
artifact bucket: mlops-nyctaxi-artifact-us-east-1-929911052415
pipeline: nyctaxi
model name: nyctaxi
workflow: arn:aws:states:us-east-1:929911052415:stateMachine:nyctaxi
```

From the AWS CodePipeline [documentation (https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-simple-s3.html)](https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-simple-s3.html):

> When Amazon S3 is the source provider for your pipeline, you may zip your source file or files into a single .zip and upload the .zip to your source bucket. You may also upload a single unzipped file; however, downstream actions that expect a .zip file will fail.

To train a model, you need multiple datasets (train, validation, and test) along with a file specifying the hyperparameters. In this example, you will create one JSON file which contains the S3 dataset locations and one JSON file which contains the hyperparameter values. Then you compress both files into a zip package to be used as input for the pipeline run.

In [13]:
```python
from io import BytesIO
import zipfile
import json

input_data = {
    'TrainingUri': s3_train_uri,
    'ValidationUri': s3_val_uri,
    'TestUri': s3_test_uri,
    'BaselineUri': s3_baseline_uri
}

hyperparameters = {
    'num_round': 50
}

zip_buffer = BytesIO()
with zipfile.ZipFile(zip_buffer, 'a') as zf:
    zf.writestr('inputData.json', json.dumps(input_data))
    zf.writestr('hyperparameters.json', json.dumps(hyperparameters))
zip_buffer.seek(0)

data_source_key = '{}/data-source.zip'.format(pipeline_name)
```

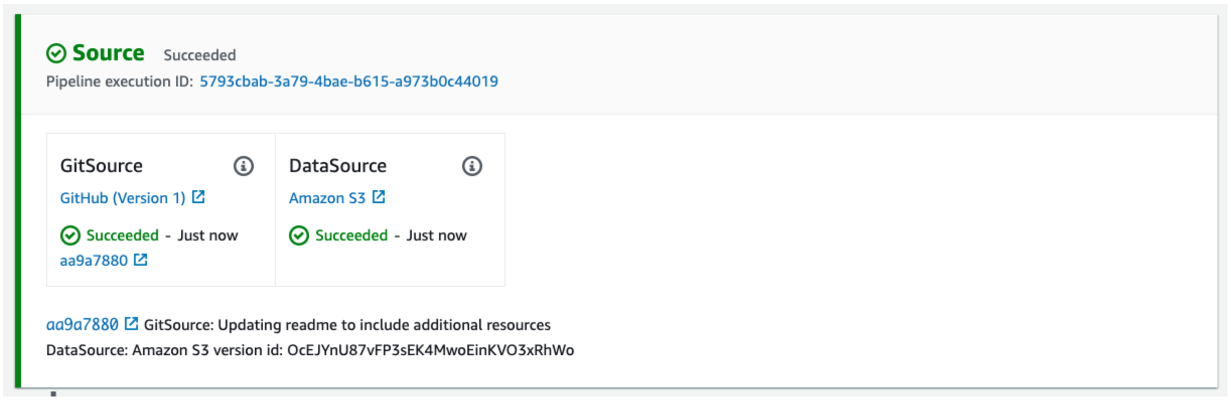
Now upload the zip package to your artifact S3 bucket - this action will trigger the pipeline to train and deploy a model.

In [14]:
```
s3 = boto3.client('s3')
s3.put_object(Bucket=artifact_bucket, Key=data_source_key, Body=bytearray(zip_bu
```

Out[14]:
```
{'ResponseMetadata': {'RequestId': 'JYP80FKSQ4X3KSTZ',
  'HostId': 'BScLPFFu+CGXWfjgzon04D+DestgOPqXM/Vyxp4v0RXQ4sTksT781vcpbkI2wq/m
I//a5Rd0IeY=',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amz-id-2': 'BScLPFFu+CGXWfjgzon04D+DestgOPqXM/Vyxp4v0RXQ4s
TksT781vcpbkI2wq/mI//a5Rd0IeY=',
   'x-amz-request-id': 'JYP80FKSQ4X3KSTZ',
   'date': 'Mon, 19 Apr 2021 06:01:06 GMT',
   'x-amz-version-id': 'xlBTA1HFMqwonpEKqdiIAHfOaETt3fNF',
   'etag': '"63b7f032efcda2347b4881094b68934e"',
   'content-length': '0',
   'server': 'AmazonS3'},
  'RetryAttempts': 0},
 'ETag': '"63b7f032efcda2347b4881094b68934e"',
 'VersionId': 'xlBTA1HFMqwonpEKqdiIAHfOaETt3fNF'}
```

Click the link below to open the AWS console at the Code Pipeline if you don't have it open in another tab.

Tip: You may need to wait a minute to see the DataSource stage turn green. The page will refresh automatically.

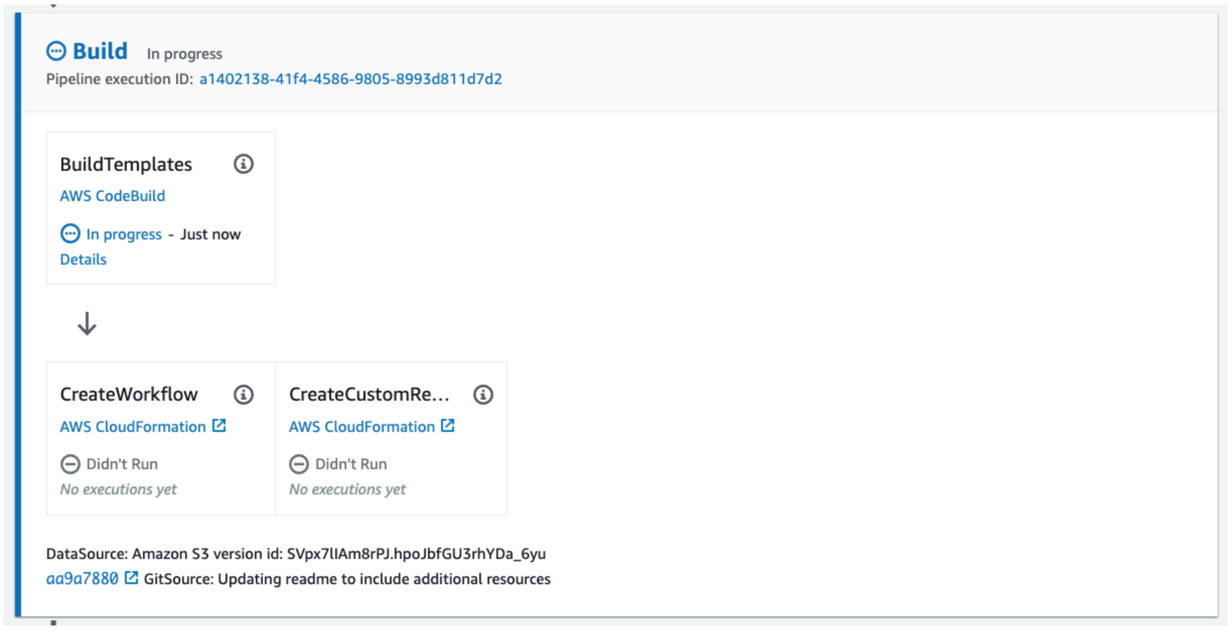

Source Succeeded
Pipeline execution ID: 5793cbab-3a79-4bae-b615-a973b0c44019
GitSource
GitHub (Version 1)
Succeeded - Just now
aa9a7880
DataSource
Amazon S3
Succeeded - Just now
aa9a7880 GitSource: Updating readme to include additional resources
DataSource: Amazon S3 version id: OcEJYnU87vFP3sEK4MwoEinKVO3xRhWo

In [15]:
```
from IPython.core.display import HTML

HTML('<a target="_blank" href="https://{0}.console.aws.amazon.com/codesuite/code
```

Out[15]: [Code Pipeline (https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/nyctaxi/view?region=us-east-1)](https://us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/nyctaxi/view?region=us-east-1)

Inspect Build Logs

Once the build stage is running, you will see the AWS CodeBuild job turn blue with a status of **In progress**.


Build In progress
Pipeline execution ID: a1402138-41f4-4586-9805-8993d811d7d2

BuildTemplates
AWS CodeBuild
In progress - Just now
Details

CreateWorkflow
AWS CloudFormation
Didn't Run
No executions yet

CreateCustomRe...
AWS CloudFormation
Didn't Run
No executions yet

DataSource: Amazon S3 version id: SVpx7lIAm8rPJ.hpoJbfGU3rhYDa_6yu
aa9a7880 GitSource: Updating readme to include additional resources

You can click on the **Details** link displayed in the CodePipeline UI or click the link below to jump directly to the CodeBuild logs.

Tip: You may need to wait a few seconds for the pipeline to transition into the active (blue) state and for the build to start.

In [53]:
```python
codepipeline = boto3.client('codepipeline')

def get_pipeline_stage(pipeline_name, stage_name):
    response = codepipeline.get_pipeline_state(name=pipeline_name)
    for stage in response['stageStates']:
        if stage['stageName'] == stage_name:
            return stage

# Get last execution id
build_stage = get_pipeline_stage(pipeline_name, 'Build')
if not 'latestExecution' in build_stage:
    raise(Exception('Please wait.  Build not started'))

build_url = build_stage['actionStates'][0]['latestExecution']['externalExecutionU

# Out a link to the code build logs
HTML('<a target="_blank" href="{0}">Code Build Logs</a>'.format(build_url))
```

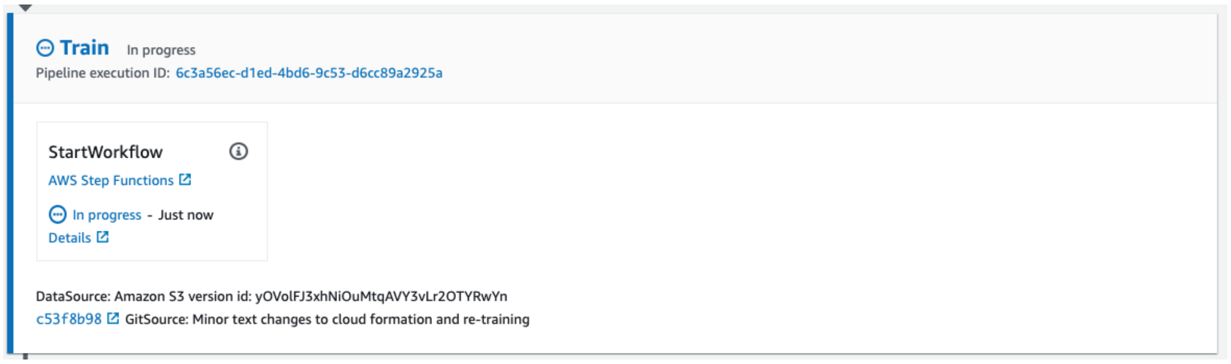
Out[53]: Code Build Logs (https://console.aws.amazon.com/codebuild/home?region=us-east-1#/builds/nyctaxi-build:1a4e3e76-eb60-4fe1-9469-a1d9b92caf2c/view/new)

The AWS CodeBuild process is responsible for creating a number of AWS CloudFormation templates which we will explore in more detail in the next section. Two of these templates are used to set up the **Train** step by creating the AWS Step Functions worklow and the custom AWS Lambda functions used within this workflow.

Train Model

Inspect Training Job

Wait until the pipeline has started running the Train step (see screenshot) before continuing with the next cells in this notebook.



DataSource: Amazon S3 version id: yOVolFJ3xhNiOuMtqAVY3vLr2OTYRwYn
c53f8b98 GitSource: Minor text changes to cloud formation and re-training

When the pipeline has started running the train step, you can click on the **Details** link displayed in the CodePipeline UI (see screenshot above) to view the Step Functions workflow which is running the training job.

Alternatively, you can click on the Workflow link from the cell output below once it's available.

In [54]:
```python
from stepfunctions.workflow import Workflow
while True:
    try:
        workflow = Workflow.attach(workflow_pipeline_arn)
        break
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)

workflow
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-54-e38083358257> in <module>
----> 1 from stepfunctions.workflow import Workflow
      2 while True:
      3     try:
      4         workflow = Workflow.attach(workflow_pipeline_arn)
      5         break

ModuleNotFoundError: No module named 'stepfunctions'
```

Or simply run the cell below to display the Step Functions workflow, and re-run it after a few minutes to see the progress.

In [18]:
```python
executions = workflow.list_executions()
if not executions:
    raise(Exception('Please wait.  Training not started'))

executions[0].render_progress()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-451defd91821> in <module>
----> 1 executions = workflow.list_executions()
      2 if not executions:
      3     raise(Exception('Please wait.  Training not started'))
      4 
      5 executions[0].render_progress()

NameError: name 'workflow' is not defined
```

Review Build Script

While you wait for the training job to complete, let's take a look at the `run.py` code which was used by the AWS CodeBuild process.

This script takes all of the input parameters, including the dataset locations and hyperparameters which you saved to JSON files earlier in this notebook, and uses them to generate the templates which the pipeline needs to run the training job. It *does not* create the actual Step Functions instance - it only generates the templates which define the Step Functions workflow, as well as the CloudFormation input templates which CodePipeline uses to instantiate the Step Functions instance.

Step-by-step, the script does the following:

1. It collects all the input parameters it needs to generate the templates. This includes information about the environment container needed to run the training job, the input and output data locations, IAM roles needed by various components, encryption keys, and more. It then sets up some basic parameters like the AWS region and the function names.
2. If the input parameters specify an environment container stored in ECR, it fetches that container. Otherwise, it fetches the URI of the AWS managed environment container needed for the training job.
3. It reads the input data JSON file which you generated earlier in this notebook (and which was included in the zip source for the pipeline), thereby fetching the locations of the train, validation, and baseline data files. Then it formats more parameters which will be needed later in the script, including version IDs and output data locations.
4. It reads the hyperparameter JSON file which you generated earlier in this notebook.
5. It defines the Step Functions workflow, starting with the input schema, followed by each step of the workflow (i.e. Create Experiment, Baseline Job, Training Job), and finally combines those steps into a workflow graph.
6. The workflow graph is saved to file, along with a file containing all of the input parameters saved according to the schema defined in the workflow.
7. It saves parameters to file which will be used by CloudFormation to instantiate the Step Functions workflow.

In [19]:
```
!pygmentize ../model/run.py
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Load parameters")
    parser.add_argument("--codebuild-id", required=True)
    parser.add_argument("--data-dir", required=True)
    parser.add_argument("--output-dir", required=True)
    parser.add_argument("--ecr-dir", required=False)
    parser.add_argument("--pipeline-name", required=True)
    parser.add_argument("--model-name", required=True)
    parser.add_argument("--deploy-role", required=True)
    parser.add_argument("--sagemaker-role", required=True)
    parser.add_argument("--sagemaker-bucket", required=True)
    parser.add_argument("--kms-key-id", required=True)
    parser.add_argument("--git-branch", required=True)
    parser.add_argument("--workflow-role-arn", required=True)
    parser.add_argument("--notification-arn", required=True)
    args = vars(parser.parse_args())
    print("args: {}".format(args))
    main(**args)
```

Customize Workflow (Optional)

If you are interested in customising the workflow used in the Build Script, store the `input_data` to be used within the local [workflow.ipynb (workflow.ipynb)](workflow.ipynb) notebook. The workflow notebook can be used to experiment with the Step Functions workflow and training job definitions for your model.

In [ ]:
```
%store input_data
```

Training Analytics

Once the training and baseline jobs are complete (meaning they are displayed in a green color in the Step Functions workflow, this takes around 5 minutes), you can inspect the experiment metrics. The code below will display all experiments in a table. Note that the baseline processing job won't have RMSE metrics - it calculates metrics based on the training data, but does not train a machine learning model.

You will explore the baseline results later in this notebook.

In [20]:
```
from sagemaker import analytics
experiment_name = 'mlops-{}'.format(model_name)
model_analytics = analytics.ExperimentAnalytics(experiment_name=experiment_name)
analytics_df = model_analytics.dataframe()

if (analytics_df.shape[0] == 0):
    raise(Exception('Please wait.  No training or baseline jobs'))

pd.set_option('display.max_colwidth', 100) # Increase column width to show full 
cols = ['TrialComponentName', 'DisplayName', 'SageMaker.InstanceType', 
        'train:rmse - Last', 'validation:rmse - Last'] # return the last rmse fo
analytics_df[analytics_df.columns & cols].head(2)
```

Out[20]:

	TrialComponentName	DisplayName	SageMaker.InstanceType	train:rmse - Last	validation:rmse - Last
0	mlops-nyctaxi-pbl-c1b29fd1-84de-4cf4-97b3-f006ceb4f717-aws-processing-job	Baseline	ml.m5.xlarge	NaN	NaN
1	mlops-nyctaxi-c1b29fd1-84de-4cf4-97b3-f006ceb4f717-aws-training-job	Training	ml.m4.xlarge	2.69577	2.74035

Deploy Dev

Test Dev Deployment

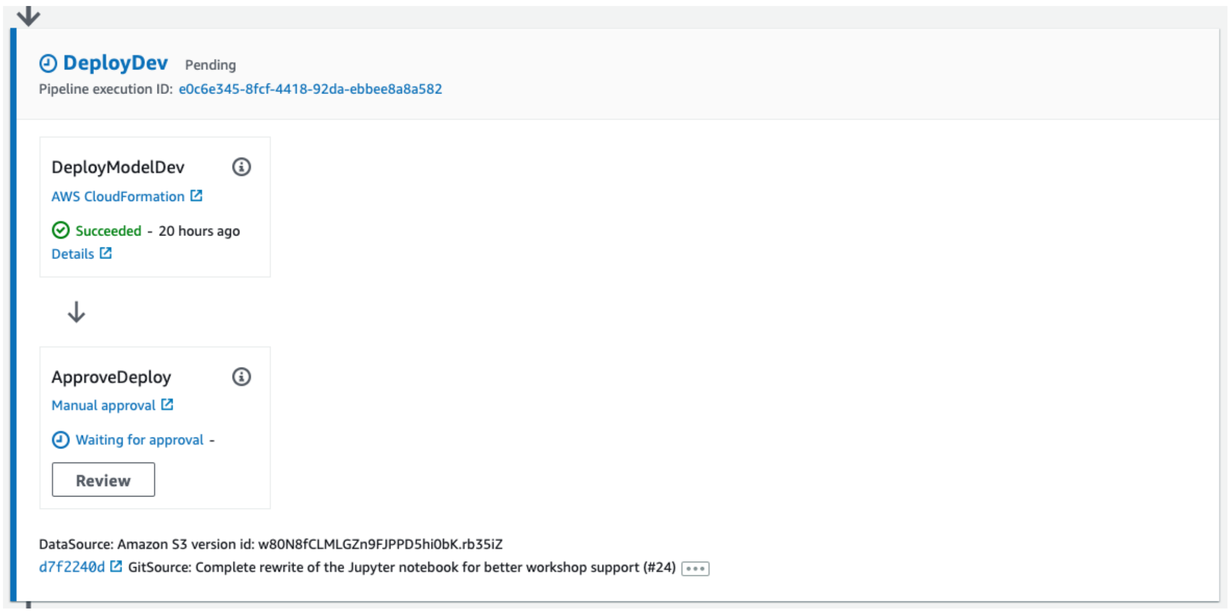
When the pipeline has finished training a model, it automatically moves to the next step, where the model is deployed as a SageMaker Endpoint. This endpoint is part of your dev deployment, therefore, in this section, you will run some tests on the endpoint to decide if you want to deploy this model into production.

First, run the cell below to fetch the name of the SageMaker Endpoint.

In [21]:
```
codepipeline = boto3.client('codepipeline')

deploy_dev = get_pipeline_stage(pipeline_name, 'DeployDev')
if not 'latestExecution' in deploy_dev:
    raise(Exception('Please wait.  Deploy dev not started'))

execution_id = deploy_dev['latestExecution']['pipelineExecutionId']
dev_endpoint_name = 'mlops-{}-dev-{}'.format(model_name, execution_id)

print('endpoint name: {}'.format(dev_endpoint_name))
```

```
---------------------------------------------------------------------------
ClientError                               Traceback (most recent call last)
<ipython-input-21-e0a725c989b2> in <module>
      1 codepipeline = boto3.client('codepipeline')
      2 
----> 3 deploy_dev = get_pipeline_stage(pipeline_name, 'DeployDev')
      4 if not 'latestExecution' in deploy_dev:
      5     raise(Exception('Please wait.  Deploy dev not started'))

<ipython-input-16-59b15dffb02f> in get_pipeline_stage(pipeline_name, stage_nam
e)
      2 
      3 def get_pipeline_stage(pipeline_name, stage_name):
----> 4     response = codepipeline.get_pipeline_state(name=pipeline_name)
      5     for stage in response['stageStates']:
      6         if stage['stageName'] == stage_name:

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api
_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operati
on_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358 
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _mak
e_api_call(self, operation_name, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ClientError: An error occurred (AccessDeniedException) when calling the GetPipe
lineState operation: User: arn:aws:sts::929911052415:assumed-role/mlops-nyctaxi
-sagemaker-role/SageMaker is not authorized to perform: codepipeline:GetPipelin
eState on resource: arn:aws:codepipeline:us-east-1:929911052415:nyctaxi
```

If you moved through the previous section very quickly, you will need to wait until the dev endpoint has been successfully deployed and the pipeline is waiting for approval to deploy to production (see screenshot). It can take up to 10 minutes for SageMaker to create an endpoint.


DeployDev Pending
Pipeline execution ID: e0c6e345-8fcf-4418-92da-ebbee8a8a582

DeployModelDev
AWS CloudFormation
Succeeded - 20 hours ago
Details

ApproveDeploy
Manual approval
Waiting for approval -
Review

DataSource: Amazon S3 version id: w80N8fCLMLGZn9FJPPD5hi0bK.rb35iZ
d7f2240d GitSource: Complete rewrite of the Jupyter notebook for better workshop support (#24)

Alternatively, run the code below to check the status of your endpoint. Wait until the status of the endpoint is 'InService'.

In [22]:
```python
sm = boto3.client('sagemaker')

while True:
    try:
        response = sm.describe_endpoint(EndpointName=dev_endpoint_name)
        print("Endpoint status: {}".format(response['EndpointStatus']))
        if response['EndpointStatus'] == 'InService':
            break
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-c3acaa9b3663> in <module>
      3 while True:
      4     try:
----> 5         response = sm.describe_endpoint(EndpointName=dev_endpoint_name)
      6         print("Endpoint status: {}".format(response['EndpointStatus']))
      7         if response['EndpointStatus'] == 'InService':

NameError: name 'dev_endpoint_name' is not defined
```

Now that your endpoint is ready, let's write some code to run the test data (which you split off from the dataset and saved to file at the start of this notebook) through the endpoint for inference. The code below supports both v1 and v2 of the SageMaker SDK, but we recommend using v2 of the SDK in all of your future projects.

In [23]:
```
import numpy as np
from tqdm import tqdm

try:
    # Support SageMaker v2 SDK: https://sagemaker.readthedocs.io/en/stable/v2.htm
    from sagemaker.predictor import Predictor
    from sagemaker.serializers import CSVSerializer
    def get_predictor(endpoint_name):
        xgb_predictor = Predictor(endpoint_name)
        xgb_predictor.serializer = CSVSerializer()
        return xgb_predictor
except:
    # Fallback to SageMaker v1.70 SDK
    from sagemaker.predictor import RealTimePredictor, csv_serializer
    def get_predictor(endpoint_name):
        xgb_predictor = RealTimePredictor(endpoint_name)
        xgb_predictor.content_type = 'text/csv'
        xgb_predictor.serializer = csv_serializer
        return xgb_predictor

def predict(predictor, data, rows=500):
    split_array = np.array_split(data, round(data.shape[0] / float(rows)))
    predictions = ''
    for array in tqdm(split_array):
        predictions = ','.join([predictions, predictor.predict(array).decode('ut
    return np.fromstring(predictions[1:], sep=',')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-23-4014f2be09a0> in <module>
      1 import numpy as np
----> 2 from tqdm import tqdm
      3 
      4 try:
      5     # Support SageMaker v2 SDK: https://sagemaker.readthedocs.io/en/sta
ble/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html)

ModuleNotFoundError: No module named 'tqdm'
```

Now use the `predict` function, which was defined in the code above, to run the test data through the endpoint and generate the predictions.

In [24]:
```
dev_predictor = get_predictor(dev_endpoint_name)
predictions = predict(dev_predictor, test_df[test_df.columns[1:]].values)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-b1451e7acb4a> in <module>
----> 1 dev_predictor = get_predictor(dev_endpoint_name)
      2 predictions = predict(dev_predictor, test_df[test_df.columns[1:]].value
s)

NameError: name 'get_predictor' is not defined
```

Next, load the predictions into a data frame, and join it with your test data. Then, calculate absolute error as the difference between the actual taxi fare and the predicted taxi fare. Display the results in a table, sorted by the highest absolute error values.

In [25]:
```
pred_df = pd.DataFrame({'total_amount_predictions': predictions })
pred_df = test_df.join(pred_df) # Join on all
pred_df['error'] = abs(pred_df['total_amount']-pred_df['total_amount_predictions

pred_df.sort_values('error', ascending=False).head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-2ecd445d630f> in <module>
----> 1 pred_df = pd.DataFrame({'total_amount_predictions': predictions })
      2 pred_df = test_df.join(pred_df) # Join on all
      3 pred_df['error'] = abs(pred_df['total_amount']-pred_df['total_amount_pr
edictions'])
      4 
      5 pred_df.sort_values('error', ascending=False).head()

NameError: name 'predictions' is not defined
```

From this table, we note that some short trip distances have large errors because the low predicted fare does not match the high actual fare. This could be the result of a generous tip which we haven't included in this dataset.

You can also analyze the results by plotting the absolute error to visualize outliers. In this graph, we see that most of the outliers are cases where the model predicted a much lower fare than the actual fare. There are only a few outliers where the model predicted a higher fare than the actual fare.

```
In [26]: sns.scatterplot(data=pred_df, x='total_amount_predictions', y='total_amount', hu
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-a345925ee420> in <module>
----> 1 sns.scatterplot(data=pred_df, x='total_amount_predictions', y='total_am
ount', hue='error')

NameError: name 'pred_df' is not defined
```

If you want one overall measure of quality for the model, you can calculate the root mean square error (RMSE) for the predicted fares compared to the actual fares. Compare this to the results calculated on the validation set at the end of the 'Inspect Training Job' section.

```
In [27]: from math import sqrt
         from sklearn.metrics import mean_squared_error

         def rmse(pred_df):
             return sqrt(mean_squared_error(pred_df['total_amount'], pred_df['total_amoun

         print('RMSE: {}'.format(rmse(pred_df)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-a32c282dfb20> in <module>
      5     return sqrt(mean_squared_error(pred_df['total_amount'], pred_df['to
tal_amount_predictions']))
      6
----> 7 print('RMSE: {}'.format(rmse(pred_df)))

NameError: name 'pred_df' is not defined
```

Deploy Prod

Approve Deployment to Production

If you are happy with the results of the model, you can go ahead and approve the model to be deployed into production. You can do so by clicking the **Review** button in the CodePipeline UI, leaving a comment to explain why you approve this model, and clicking on **Approve**.

Alternatively, you can create a Jupyter widget which (when enabled) allows you to comment and approve the model directly from this notebook. Run the cell below to see this in action.

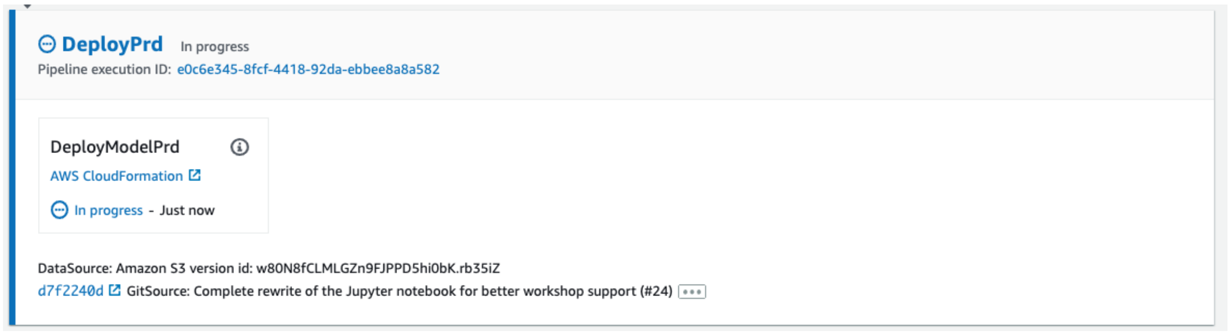
In [28]:
```
import ipywidgets as widgets

def on_click(obj):
    result = { 'summary': approval_text.value, 'status': obj.description }
    response = codepipeline.put_approval_result(
      pipelineName=pipeline_name,
      stageName='DeployDev',
      actionName='ApproveDeploy',
      result=result,
      token=approval_action['token']
    )
    button_box.close()
    print(result)

# Create the widget if we are ready for approval
deploy_dev = get_pipeline_stage(pipeline_name, 'DeployDev')
if not 'latestExecution' in deploy_dev['actionStates'][-1]:
    raise(Exception('Please wait.  Deploy dev not complete'))

approval_action = deploy_dev['actionStates'][-1]['latestExecution']
if approval_action['status'] == 'Succeeded':
    print('Dev approved: {}'.format(approval_action['summary']))
elif 'token' in approval_action:
    approval_text = widgets.Text(placeholder='Optional approval message')
    approve_btn = widgets.Button(description="Approved", button_style='success',
    reject_btn = widgets.Button(description="Rejected", button_style='danger', i
    approve_btn.on_click(on_click)
    reject_btn.on_click(on_click)
    button_box = widgets.HBox([approval_text, approve_btn, reject_btn])
    display(button_box)
else:
    raise(Exception('Please wait. No dev approval'))
```

```
---------------------------------------------------------------------------
ClientError                               Traceback (most recent call last)
<ipython-input-28-550982b66e23> in <module>
     14 
     15 # Create the widget if we are ready for approval
---> 16 deploy_dev = get_pipeline_stage(pipeline_name, 'DeployDev')
     17 if not 'latestExecution' in deploy_dev['actionStates'][-1]:
     18     raise(Exception('Please wait.  Deploy dev not complete'))

<ipython-input-16-59b15dffb02f> in get_pipeline_stage(pipeline_name, stage_na
me)
      2 
      3 def get_pipeline_stage(pipeline_name, stage_name):
----> 4     response = codepipeline.get_pipeline_state(name=pipeline_name)
      5     for stage in response['stageStates']:
      6         if stage['stageName'] == stage_name:

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _a
pi_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_opera
tion_name)
    356             # The "self" in this scope is referring to the BaseClien
t.
```

```
--> 357                 return self._make_api_call(operation_name, kwargs)
    358 
    359             _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _m
ake_api_call(self, operation_name, api_params)
    674                 error_code = parsed_response.get("Error", {}).get("Code")
    675                 error_class = self.exceptions.from_code(error_code)
--> 676                 raise error_class(parsed_response, operation_name)
    677             else:
    678                 return parsed_response

ClientError: An error occurred (AccessDeniedException) when calling the GetPi
pelineState operation: User: arn:aws:sts::929911052415:assumed-role/mlops-nyc
taxi-sagemaker-role/SageMaker is not authorized to perform: codepipeline:GetP
ipelineState on resource: arn:aws:codepipeline:us-east-1:929911052415:nyctaxi
```

Test Production Deployment

Within about a minute after approving the model deployment, you should see the pipeline start on the final step: deploying your model into production. In this section, you will check the deployment status and test the production endpoint after it has been deployed.

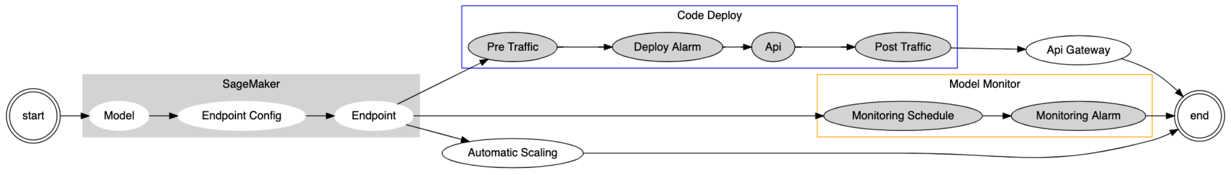

DeployPrd In progress
Pipeline execution ID: e0c6e345-8fcf-4418-92da-ebbee8a8a582

DeployModelPrd
AWS CloudFormation
In progress - Just now

DataSource: Amazon S3 version id: w80N8fCLMLGZn9FJPPD5hi0bK.rb35iZ
d7f2240d GitSource: Complete rewrite of the Jupyter notebook for better workshop support (#24)

This step of the pipeline uses CloudFormation to deploy a number of resources on your behalf. In particular, it creates:

1. A production-ready SageMaker Endpoint for your model, with data capture (https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-data-capture.html) (used by SageMaker Model Monitor) and autoscaling (https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html) enabled.
2. A model monitoring schedule (https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-scheduling.html) which outputs the results to CloudWatch metrics, along with a CloudWatch Alarm (https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.htn which will notify you when a violation occurs.
3. A CodeDeploy instance which creates a simple app by deploying API Gateway, three Lambda functions, and an alarm to notify of the success or failure of this deployment. The code for the Lambda functions can be found in `api/app.py` , `api/pre_traffic_hook.py` , and `api/post_traffic_hook.py` . These functions update the endpoint to enable data capture, format and submit incoming traffic to the SageMaker endpoint, and capture the data logs.



Let's check how the deployment is progressing. Use the code below to fetch the execution ID of the depoyment step. Then generate a table which lists the resources created by the CloudFormation stack and their creation status. You can re-run the cell after a few minutes to see how the steps are progressing.

In [30]:
```
deploy_prd = get_pipeline_stage(pipeline_name, 'DeployPrd')
if not 'latestExecution' in deploy_prd or not 'latestExecution' in deploy_prd['a
    raise(Exception('Please wait.  Deploy prd not started'))

execution_id = deploy_prd['latestExecution']['pipelineExecutionId']
```

```
---------------------------------------------------------------------------
ClientError                               Traceback (most recent call last)
<ipython-input-30-c3435655f877> in <module>
----> 1 deploy_prd = get_pipeline_stage(pipeline_name, 'DeployPrd')
      2 if not 'latestExecution' in deploy_prd or not 'latestExecution' in depl
oy_prd['actionStates'][0]:
      3     raise(Exception('Please wait.  Deploy prd not started'))
      4 
      5 execution_id = deploy_prd['latestExecution']['pipelineExecutionId']

<ipython-input-16-59b15dffb02f> in get_pipeline_stage(pipeline_name, stage_nam
e)
      2 
      3 def get_pipeline_stage(pipeline_name, stage_name):
----> 4     response = codepipeline.get_pipeline_state(name=pipeline_name)
      5     for stage in response['stageStates']:
      6         if stage['stageName'] == stage_name:

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api
_call(self, *args, **kwargs)
    355                         "%s() only accepts keyword arguments." % py_operati
on_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358 
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _mak
e_api_call(self, operation_name, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ClientError: An error occurred (AccessDeniedException) when calling the GetPipe
lineState operation: User: arn:aws:sts::929911052415:assumed-role/mlops-nyctaxi
-sagemaker-role/SageMaker is not authorized to perform: codepipeline:GetPipelin
eState on resource: arn:aws:codepipeline:us-east-1:929911052415:nyctaxi
```

In [31]:
```
from datetime import datetime, timedelta
from dateutil.tz import tzlocal

def get_event_dataframe(events):
    stack_cols = ['LogicalResourceId', 'ResourceStatus', 'ResourceStatusReason',
    stack_event_df = pd.DataFrame(events)[stack_cols].fillna('')
    stack_event_df['TimeAgo'] = (datetime.now(tzlocal())-stack_event_df['Timesta
    return stack_event_df.drop('Timestamp', axis=1)

cfn = boto3.client('cloudformation')

stack_name = stack_name='{}-deploy-prd'.format(pipeline_name)
print('stack name: {}'.format(stack_name))

# Get latest stack events
while True:
    try:
        response = cfn.describe_stack_events(StackName=stack_name)
        break
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)

get_event_dataframe(response['StackEvents']).head()
```

```
stack name: nyctaxi-deploy-prd
```

Out[31]:

	LogicalResourceId	ResourceStatus	ResourceStatusReason	TimeAgo
0	nyctaxi-deploy-prd	CREATE_COMPLETE		0 days 01:58:44.244434
1	ApiFunctionInvokePermissionProd	CREATE_COMPLETE		0 days 01:58:46.399434
2	ServerlessRestApiProdStage	CREATE_COMPLETE		0 days 01:58:52.486434
3	ServerlessRestApiProdStage	CREATE_IN_PROGRESS	Resource creation Initiated	0 days 01:58:53.205434
4	ServerlessRestApiProdStage	CREATE_IN_PROGRESS		0 days 01:58:54.107434

The resource of most interest to us is the endpoint. This takes on average 10 minutes to deploy. In the meantime, you can take a look at the Python code used for the application.

The `app.py` is the main entry point invoking the Amazon SageMaker endpoint. It returns results along with a custom header for the endpoint we invoked.

In [32]:
```
!pygmentize ../api/app.py
```

```
import json
import logging
import os

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

sm_runtime = boto3.client("sagemaker-runtime")


def lambda_handler(event, context):
    logger.debug("event %s", json.dumps(event))
    endpoint_name = os.environ["ENDPOINT_NAME"]
    logger.info("api for endpoint %s", endpoint_name)

    # Get posted body and content type
    content_type = event["headers"].get("Content-Type", "text/csv")
    custom_attributes = event["headers"].get("X-Amzn-SageMaker-Custom-Attribu
tes", "")
    if content_type.startswith("text/csv"):
        payload = event["body"]
    elif content_type.startswith("application/json"):
        payload = json.loads(event["body"])
    else:
        message = "bad content type: {}".format(content_type)
        logger.error()
        return {"statusCode": 500, "message": message}

    logger.info("content type: %s size: %d", content_type, len(payload))

    try:
        # Invoke the endpoint with full multi-line payload
        response = sm_runtime.invoke_endpoint(
            EndpointName=endpoint_name,
            Body=payload,
            ContentType=content_type,
            CustomAttributes=custom_attributes,
            Accept="application/json",
        )
        # Return predictions as JSON dictionary instead of CSV text
        predictions = response["Body"].read().decode("utf-8")
        return {
            "statusCode": 200,
            "headers": {
                "Content-Type": content_type,
                "X-SageMaker-Endpoint": endpoint_name,
            },
            "body": predictions,
        }
    except ClientError as e:
        logger.error(
            "Unexpected sagemaker error: {}".format(e.response["Error"]["Mess
age"])
        )
        logger.error(e)
        return {"statusCode": 500, "message": "Unexpected sagemaker error"}
```

The `pre_traffic_hook.py` lambda is invoked prior to deployment and confirms the endpoint has data capture enabled.

In [33]:
```
!pygmentize ../api/pre_traffic_hook.py
```

```
import json
import logging
import os

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

sm = boto3.client("sagemaker")
cd = boto3.client("codedeploy")


def lambda_handler(event, context):
    logger.debug("event %s", json.dumps(event))
    endpoint_name = os.environ["ENDPOINT_NAME"]
    logger.info("pre traffic for endpoint %s", endpoint_name)

    error_message = None
    try:
        # Update endpoint to enable data capture
        response = sm.describe_endpoint(EndpointName=endpoint_name)
        status = response["EndpointStatus"]
        if status != "InService":
            error_message = "SageMaker endpoint: {} status: {} not InService"
.format(
                endpoint_name, status
            )
        else:
            # Validate that endpoint config has data capture enabled
            endpoint_config_name = response["EndpointConfigName"]
            response = sm.describe_endpoint_config(
                EndpointConfigName=endpoint_config_name
            )
            if (
                "DataCaptureConfig" in response
                and response["DataCaptureConfig"]["EnableCapture"]
            ):
                logger.info(
                    "data capture enabled for endpoint config %s", endpoint_c
onfig_name
                )
            else:
                error_message = "SageMaker data capture not enabled for endpo
int config"
            # TODO: Invoke endpoint if don't have canary / live traffic
    except ClientError as e:
        error_message = e.response["Error"]["Message"]
        logger.error("Error checking endpoint %s", error_message)

    try:
        if error_message != None:
            logger.info("put codepipeline failed: %s", error_message)
```

```
                response = cd.put_lifecycle_event_hook_execution_status(
                    deploymentId=event["DeploymentId"],
                    lifecycleEventHookExecutionId=event["LifecycleEventHookExecut
ionId"],
                    status="Failed",
                )
                return {"statusCode": 400, "message": error_message}
            else:
                logger.info("put codepipeline success")
                response = cd.put_lifecycle_event_hook_execution_status(
                    deploymentId=event["DeploymentId"],
                    lifecycleEventHookExecutionId=event["LifecycleEventHookExecut
ionId"],
                    status="Succeeded",
                )
                return {
                    "statusCode": 200,
                }
        except ClientError as e:
            # Error attempting to update the cloud formation
            logger.error("Unexpected codepipeline error")
            logger.error(e)
            return {"statusCode": 500, "message": e.response["Error"]["Message"]}
```

The `post_traffic_hook.py` lambda is invoked to perform any final checks, in this case to verify that we have received log data from data capature.

In [34]:
```
!pygmentize ../api/post_traffic_hook.py
```

```
import json
import logging
import os

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

sm = boto3.client("sagemaker")
s3 = boto3.client("s3")
cd = boto3.client("codedeploy")


def get_bucket_prefix(url):
    try:
        from urllib.parse import urlparse
    except ImportError:
        from urlparse import urlparse
    a = urlparse(url)
    return a.netloc, a.path.lstrip("/") + "/"


def lambda_handler(event, context):
    logger.debug("event %s", json.dumps(event))
    endpoint_name = os.environ["ENDPOINT_NAME"]
    logger.info("post traffic for endpoint: %s", endpoint_name)
    data_capture_uri = os.environ.get("DATA_CAPTURE_URI", "")
    logger.info("data capture uri: %s", data_capture_uri)

    error_message = None
    try:
        if data_capture_uri:
            # List objects under data capture logs director
            bucket, prefix = get_bucket_prefix(data_capture_uri)
            contents = s3.list_objects(Bucket=bucket, Prefix=prefix).get("Con
tents")
            if contents != None and contents:
                logger.info("Found %d data capture logs", len(contents))
            else:
                error_message = "No data capture logs found"
                logger.error(error_message)
    except ClientError as e:
        error_message = e.response["Error"]["Message"]
        logger.error("Error checking logs %s", error_message)

    try:
        if error_message != None:
            logger.info("put codepipeline failed: %s", error_message)
            response = cd.put_lifecycle_event_hook_execution_status(
                deploymentId=event["DeploymentId"],
                lifecycleEventHookExecutionId=event["LifecycleEventHookExecut
ionId"],
                status="Failed",
            )
            return {"statusCode": 400, "message": error_message}
        else:
            logger.info("put codepipeline success")
            response = cd.put_lifecycle_event_hook_execution_status(
                deploymentId=event["DeploymentId"],
                lifecycleEventHookExecutionId=event["LifecycleEventHookExecut
ionId"],
                status="Succeeded",
            )
            return {
                "statusCode": 200,
            }
    except ClientError as e:
        # Error attempting to update the cloud formation
        logger.error("Unexpected codepipeline error")
        logger.error(e)
        return {"statusCode": 500, "message": e.response["Error"]["Message"]}
```

Use the code below to fetch the name of the endpoint, then run a loop to wait for the endpoint to be fully deployed. You need the status to be 'InService'.

In [35]:
```
prd_endpoint_name='mlops-{}-prd-{}'.format(model_name, execution_id)
print('prod endpoint: {}'.format(prd_endpoint_name))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-622dfcdb8c48> in <module>
----> 1 prd_endpoint_name='mlops-{}-prd-{}'.format(model_name, execution_id)
      2 print('prod endpoint: {}'.format(prd_endpoint_name))

NameError: name 'execution_id' is not defined
```

In [36]:
```python
sm = boto3.client('sagemaker')

while True:
    try:
        response = sm.describe_endpoint(EndpointName=prd_endpoint_name)
        print("Endpoint status: {}".format(response['EndpointStatus']))
        # Wait until the endpoint is in service with data capture enabled
        if response['EndpointStatus'] == 'InService' \
            and 'DataCaptureConfig' in response \
            and response['DataCaptureConfig']['EnableCapture']:
            break
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-24f5cbd1ad9a> in <module>
      3 while True:
      4     try:
----> 5         response = sm.describe_endpoint(EndpointName=prd_endpoint_name)
      6         print("Endpoint status: {}".format(response['EndpointStatus']))
      7         # Wait until the endpoint is in service with data capture enabl
ed

NameError: name 'prd_endpoint_name' is not defined
```

When the endpoint status is 'InService', you can continue. Earlier in this notebook, you created some code to send data to the dev endpoint. Reuse this code now to send a sample of the test data to the production endpoint. Since data capture is enabled on this endpoint, you want to send single records at a time, so the model monitor can map these records to the baseline.

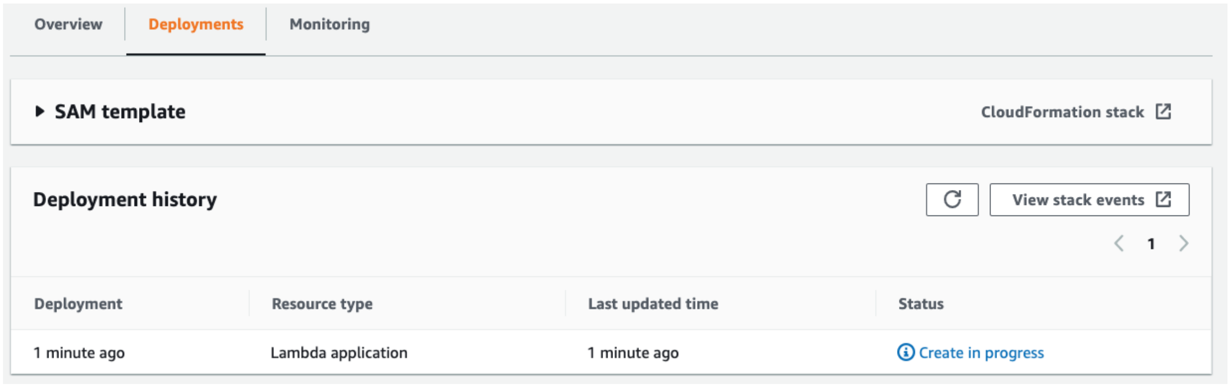
You will inspect the model monitor later in this notebook. For now, just check if you can send data to the endpoint and receive predictions in return.

In [37]:
```python
prd_predictor = get_predictor(prd_endpoint_name)
sample_values = test_df[test_df.columns[1:]].sample(100).values
predictions = predict(prd_predictor, sample_values, rows=1)
predictions
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-31d2886a8554> in <module>
----> 1 prd_predictor = get_predictor(prd_endpoint_name)
      2 sample_values = test_df[test_df.columns[1:]].sample(100).values
      3 predictions = predict(prd_predictor, sample_values, rows=1)
      4 predictions

NameError: name 'get_predictor' is not defined
```

Test REST API

Although you already tested the SageMaker endpoint in the previous section, it is also a good idea to test the application created with API Gateway.



Deployment	Resource type	Last updated time	Status
1 minute ago	Lambda application	1 minute ago	Create in progress

Follow the link below to open the Lambda Deployment where you can see the in-progress and completed deployments. You can also click to expand the **SAM template** to see the packaged CloudFormation template used in the deployment.

In [38]:
```
HTML('<a target="_blank" href="https://{0}.console.aws.amazon.com/lambda/home?reg
```

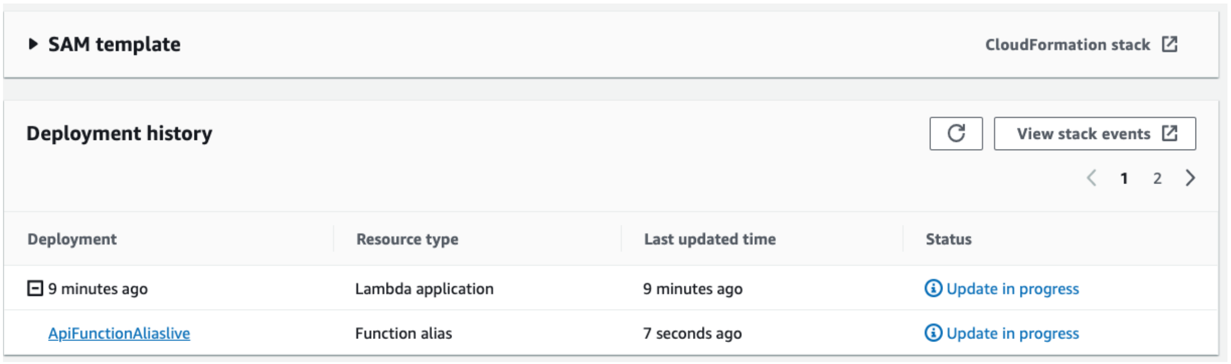
Out[38]: Lambda Deployment (https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/applications/nyctaxi-deploy-prd?tab=deploy)

Run the code below to confirm that the endpoint is in service. It will complete once the REST API is available.

In [39]:
```
def get_stack_status(stack_name):
    response = cfn.describe_stacks(StackName=stack_name)
    if response['Stacks']:
        stack = response['Stacks'][0]
        outputs = None
        if 'Outputs' in stack:
            outputs = dict([(o['OutputKey'], o['OutputValue']) for o in stack['O
        return stack['StackStatus'], outputs

outputs = None
while True:
    try:
        status, outputs = get_stack_status(stack_name)
        response = sm.describe_endpoint(EndpointName=prd_endpoint_name)
        print("Endpoint status: {}".format(response['EndpointStatus']))
        if outputs:
            break
        elif status.endswith('FAILED'):
            raise(Exception('Stack status: {}'.format(status)))
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)

if outputs:
    print('deployment application: {}'.format(outputs['DeploymentApplication']))
    print('rest api: {}'.format(outputs['RestApi']))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-39-a83f5350f733> in <module>
     12     try:
     13         status, outputs = get_stack_status(stack_name)
---> 14         response = sm.describe_endpoint(EndpointName=prd_endpoint_name)
     15         print("Endpoint status: {}".format(response['EndpointStatus']))
     16         if outputs:

NameError: name 'prd_endpoint_name' is not defined
```

If you are performing an update on your production deployment as a result of running Trigger Retraining you will then be able to expand the Lambda Deployment tab to reveal the resources. Click on the **ApiFunctionAliaslive** link to see the Lambda Deployment in progress.



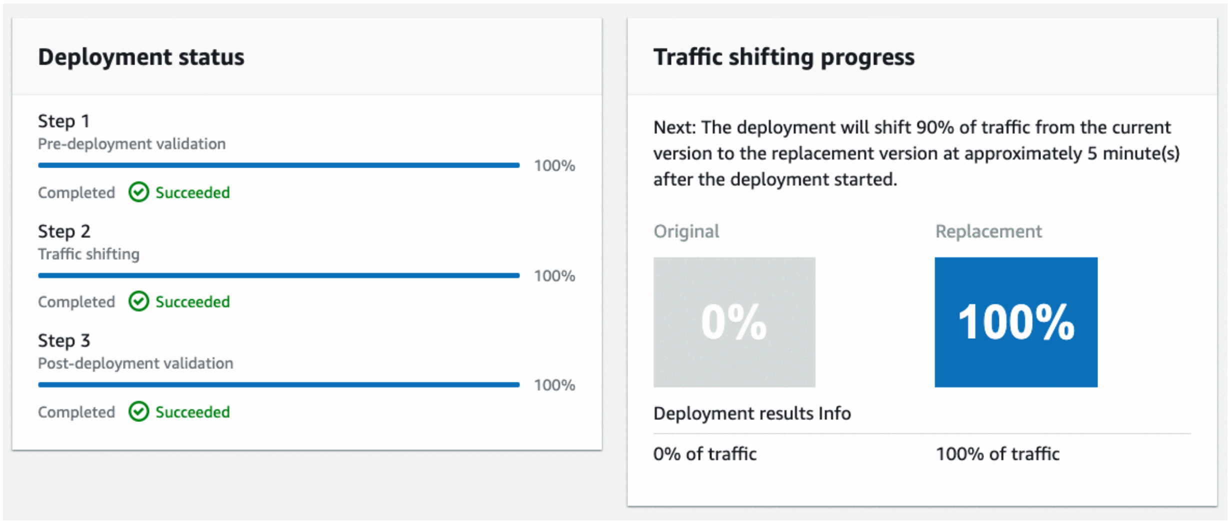
Deployment	Resource type	Last updated time	Status
9 minutes ago	Lambda application	9 minutes ago	Update in progress
ApiFunctionAliaslive	Function alias	7 seconds ago	Update in progress

This page will be updated to list the deployment events. It also has a link to the Deployment Application which you can access in the output of the next cell.

In [40]:
```
HTML('<a target="_blank" href="https://{0}.console.aws.amazon.com/codesuite/codec
```

Out[40]: [CodeDeploy application (https://us-east-1.console.aws.amazon.com/codesuite/codedeploy/applications/nyctaxi-deploy-prd-ServerlessDeploymentApplication-1T0VKOI6MKRFS?region=us-east-1)](https://us-east-1.console.aws.amazon.com/codesuite/codedeploy/applications/nyctaxi-deploy-prd-ServerlessDeploymentApplication-1T0VKOI6MKRFS?region=us-east-1)

CodeDeploy will perform a canary deployment and send 10% of the traffic to the new endpoint over a 5-minute period.


Deployment status

Step 1
Pre-deployment validation
100%
Completed ✓ Succeeded

Step 2
Traffic shifting
100%
Completed ✓ Succeeded

Step 3
Post-deployment validation
100%
Completed ✓ Succeeded

Traffic shifting progress

Next: The deployment will shift 90% of traffic from the current version to the replacement version at approximately 5 minute(s) after the deployment started.

Original	Replacement
0%	100%

Deployment results Info

0% of traffic	100% of traffic

We can invoke the REST API and inspect the headers being returned to see which endpoint we are hitting. You will occasionally see the cell below show a different endpoint that settles to the new version once the stack is complete.

In [41]:
```
%%time

from urllib import request

headers = {"Content-type": "text/csv"}
payload = test_df[test_df.columns[1:]].head(1).to_csv(header=False, index=False)
rest_api = outputs['RestApi']

while True:
    try:
        resp = request.urlopen(request.Request(rest_api, data=payload, headers=he
        print("Response code: %d: endpoint: %s" % (resp.getcode(), resp.getheader
        status, outputs = get_stack_status(stack_name)
        if status.endswith('COMPLETE'):
            print('Deployment complete\n')
            break
        elif status.endswith('FAILED'):
            raise(Exception('Stack status: {}'.format(status)))
    except ClientError as e:
        print(e.response["Error"]["Message"])
    time.sleep(10)
```

```
Response code: 200: endpoint: mlops-nyctaxi-prd-c1b29fd1-84de-4cf4-97b3-f006ceb
4f717
Deployment complete

CPU times: user 26.3 ms, sys: 3.34 ms, total: 29.6 ms
Wall time: 966 ms
```

Monitor

Inspect Model Monitor

When you prepared the datasets for model training at the start of this notebook, you saved a baseline dataset (a copy of the train dataset). Then, when you approved the model for deployment into production, the pipeline set up an SageMaker Endpoint with data capture enabled and a model monitoring schedule. In this section, you will take a closer look at the model monitor results.

To start off, fetch the latest production deployment execution ID.

In [42]:
```
deploy_prd = get_pipeline_stage(pipeline_name, 'DeployPrd')
if not 'latestExecution' in deploy_prd:
    raise(Exception('Please wait.  Deploy prod not complete'))

execution_id = deploy_prd['latestExecution']['pipelineExecutionId']
```

```
---------------------------------------------------------------------------
ClientError                               Traceback (most recent call last)
<ipython-input-42-c6464989157b> in <module>
----> 1 deploy_prd = get_pipeline_stage(pipeline_name, 'DeployPrd')
      2 if not 'latestExecution' in deploy_prd:
      3     raise(Exception('Please wait.  Deploy prod not complete'))
      4 
      5 execution_id = deploy_prd['latestExecution']['pipelineExecutionId']

<ipython-input-16-59b15dffb02f> in get_pipeline_stage(pipeline_name, stage_nam
e)
      2 
      3 def get_pipeline_stage(pipeline_name, stage_name):
----> 4     response = codepipeline.get_pipeline_state(name=pipeline_name)
      5     for stage in response['stageStates']:
      6         if stage['stageName'] == stage_name:

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api
_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operati
on_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358 
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _mak
e_api_call(self, operation_name, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ClientError: An error occurred (AccessDeniedException) when calling the GetPipe
lineState operation: User: arn:aws:sts::929911052415:assumed-role/mlops-nyctaxi
-sagemaker-role/SageMaker is not authorized to perform: codepipeline:GetPipelin
eState on resource: arn:aws:codepipeline:us-east-1:929911052415:nyctaxi
```

Under the hood, SageMaker model monitor runs in SageMaker processing jobs. Use the execution ID to fetch the names of the processing job and the schedule.

In [43]:
```
processing_job_name='mlops-{}-pbl-{}'.format(model_name, execution_id)
schedule_name='mlops-{}-pms'.format(model_name)

print('processing job name: {}'.format(processing_job_name))
print('schedule name: {}'.format(schedule_name))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-43-c6dbf919f892> in <module>
----> 1 processing_job_name='mlops-{}-pbl-{}'.format(model_name, execution_id)
      2 schedule_name='mlops-{}-pms'.format(model_name)
      3 
      4 print('processing job name: {}'.format(processing_job_name))
      5 print('schedule name: {}'.format(schedule_name))

NameError: name 'execution_id' is not defined
```

Explore Baseline

Now fetch the baseline results from the processing job. This cell will throw an exception if the processing job is not complete - if that happens, just wait several minutes and try again.

In [44]:
```
import sagemaker
from sagemaker.model_monitor import BaseliningJob, MonitoringExecution
from sagemaker.s3 import S3Downloader

sagemaker_session = sagemaker.Session()
baseline_job = BaseliningJob.from_processing_name(sagemaker_session, processing_j
status = baseline_job.describe()['ProcessingJobStatus']
if status != 'Completed':
    raise(Exception('Please wait. Processing job not complete, status: {}'.forma

baseline_results_uri  = baseline_job.outputs[0].destination
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-44-45ab6f0c072f> in <module>
      4 
      5 sagemaker_session = sagemaker.Session()
----> 6 baseline_job = BaseliningJob.from_processing_name(sagemaker_session, pr
ocessing_job_name)
      7 status = baseline_job.describe()['ProcessingJobStatus']
      8 if status != 'Completed':

NameError: name 'processing_job_name' is not defined
```

SageMaker model monitor generates two types of files. Take a look at the statistics file first. It calculates various statistics for each feature of the dataset, including the mean, standard deviation, minimum value, maximum value, and more.

In [45]:
```
import pandas as pd
import json

baseline_statistics = baseline_job.baseline_statistics().body_dict
schema_df = pd.json_normalize(baseline_statistics["features"])
schema_df[["name", "numerical_statistics.mean", "numerical_statistics.std_dev",
           "numerical_statistics.min", "numerical_statistics.max"]].head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-45-c6f61f40b902> in <module>
      2 import json
      3 
----> 4 baseline_statistics = baseline_job.baseline_statistics().body_dict
      5 schema_df = pd.json_normalize(baseline_statistics["features"])
      6 schema_df[["name", "numerical_statistics.mean", "numerical_statistics.s
td_dev",

NameError: name 'baseline_job' is not defined
```

Now look at the suggested constraints files (https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-byoc-constraints.html). As the name implies, these are constraints which SageMaker model monitor recommends. If the live data which is sent to your production SageMaker Endpoint violates these constraints, this indicates data drift, and model monitor can raise an alert to trigger retraining. Of course, you can set different constraints based on the statistics which you viewed previously.

In [46]:
```
baseline_constraints = baseline_job.suggested_constraints().body_dict
constraints_df = pd.json_normalize(baseline_constraints["features"])
constraints_df.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-69609ac64009> in <module>
----> 1 baseline_constraints = baseline_job.suggested_constraints().body_dict
      2 constraints_df = pd.json_normalize(baseline_constraints["features"])
      3 constraints_df.head()

NameError: name 'baseline_job' is not defined
```

View data capture

When the "Deploy Production" stage of the MLOps pipeline deploys a SageMaker endpoint, it also enables data capture. This means the incoming requests to the endpoint, as well as the results from the ML model, are stored in an S3 location. Model monitor can analyze this data and compare it to the baseline to ensure that no constraints are violated.

Use the code below to check how many files have been created by the data capture, and view the latest file in detail. Note, data capture relies on data being sent to the production endpoint. If you don't see any files yet, wait several minutes and try again.

In [47]:
```
bucket = sagemaker_session.default_bucket()
data_capture_logs_uri = 's3://{}/{}/datacapture/{}'.format(bucket, model_name, p

capture_files = S3Downloader.list(data_capture_logs_uri)
print('Found {} files'.format(len(capture_files)))

if capture_files:
    # Get the first line of the most recent file
    event = json.loads(S3Downloader.read_file(capture_files[-1]).split('\n')[0])
    print('\nLast file:\n{}'.format(json.dumps(event, indent=2)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-e80323c9bd6a> in <module>
      1 bucket = sagemaker_session.default_bucket()
----> 2 data_capture_logs_uri = 's3://{}/{}/datacapture/{}'.format(bucket, mode
l_name, prd_endpoint_name)
      3 
      4 capture_files = S3Downloader.list(data_capture_logs_uri)
      5 print('Found {} files'.format(len(capture_files)))

NameError: name 'prd_endpoint_name' is not defined
```

View monitoring schedule

There are some useful functions for plotting and rendering distribution statistics or constraint violations provided in a `utils` file in the SageMaker Examples GitHub (https://github.com/aws/amazon-sagemaker-examples/tree/master/sagemaker_model_monitor/visualization). Grab a copy of this code to use in this notebook.

In [48]:
```
!wget -O utils.py --quiet https://raw.githubusercontent.com/awslabs/amazon-sagem
import utils as mu
```

The minimum scheduled run time (https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-scheduling.html) for model monitor is one hour, which means you will need to wait at least an hour to see any results. Use the code below to check the schedule status and list the next run. If you are completing this notebook as part of a workshop, your host will have activities which you can complete while you wait.

In [49]:
```
sm = boto3.client('sagemaker')

response = sm.describe_monitoring_schedule(MonitoringScheduleName=schedule_name)
print('Schedule Status: {}'.format(response['MonitoringScheduleStatus']))

now = datetime.now(tzlocal())
next_hour = (now+timedelta(hours=1)).replace(minute=0)
scheduled_diff = (next_hour-now).seconds//60
print('Next schedule in {} minutes'.format(scheduled_diff))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-49-3cb2287e6be1> in <module>
      1 sm = boto3.client('sagemaker')
      2 
----> 3 response = sm.describe_monitoring_schedule(MonitoringScheduleName=sched
ule_name)
      4 print('Schedule Status: {}'.format(response['MonitoringScheduleStatus']
))
      5 

NameError: name 'schedule_name' is not defined
```

While you wait, you can take a look at the CloudFormation template which is used as a base for the CloudFormation template built by CodeDeploy to deploy the production application.

Alterntively, you can jump ahead to Trigger Retraining which will kick off another run of the code pipeline whilst you wait.

In [50]:
```
!cat ../assets/deploy-model-prd.yml
```

```
          - InitialInstanceCount: 2
            InitialVariantWeight: 1.0
            InstanceType: ml.m5.large

            ModelName: !GetAtt Model.ModelName
            VariantName: !Sub ${ModelVariant}-${ModelName}
        DataCaptureConfig:
          CaptureContentTypeHeader:
            CsvContentTypes:
              - "text/csv"
            JsonContentTypes:
              - "application/json"
          CaptureOptions:
            - CaptureMode: Input
            - CaptureMode: Output
          DestinationS3Uri: !Sub s3://sagemaker-${AWS::Region}-${AWS::AccountI
d}/${ModelName}/datacapture
          EnableCapture: True
          InitialSamplingPercentage: 100
          KmsKeyId: !Ref KmsKeyId
        EndpointConfigName: !Sub mlops-${ModelName}-pec-${TrainJobId}
```

A couple of minutes after the model monitoring schedule has run, you can use the code below to fetch the latest schedule status. A completed schedule run may have found violations.

In [ ]:
```
processing_job_arn = None

while processing_job_arn == None:
    try:
        response = sm.list_monitoring_executions(MonitoringScheduleName=schedule_
    except ClientError as e:
        print(e.response["Error"]["Message"])
    for mon in response['MonitoringExecutionSummaries']:
        status = mon['MonitoringExecutionStatus']
        now = datetime.now(tzlocal())
        created_diff = (now-mon['CreationTime']).seconds//60
        print('Schedule status: {}, Created: {} minutes ago'.format(status, crea
        if status in ['Completed', 'CompletedWithViolations']:
            processing_job_arn = mon['ProcessingJobArn']
            break
        if status == 'InProgress':
            break
    else:
        raise(Exception('Please wait.  No Schedules executing'))
    time.sleep(10)
```

View monitoring results

Once the model monitoring schedule has had a chance to run at least once, you can take a look at the results. First, load the monitoring execution results from the latest scheduled run.

In [ ]:
```
if processing_job_arn:
    execution = MonitoringExecution.from_processing_arn(sagemaker_session=sagemal
                                                        processing_job_arn=proce
    exec_inputs = {inp['InputName']: inp for inp in execution.describe()['Proces
    exec_results_uri = execution.output.destination

    print('Monitoring Execution results: {}'.format(exec_results_uri))
```

Take a look at the files which have been saved in the S3 output location. If violations were found, you should see a constraint violations file in addition to the statistics and constraints file which you viewed before.

In [ ]:
```
!aws s3 ls $exec_results_uri/
```

Now, fetch the monitoring statistics and violations. Then use the utils code to visualize the results in a table. It will highlight any baseline drift found by the model monitor. Drift can happen for categorical features (for inferred string styles) or for numerical features (e.g. total fare amount).

In [ ]:
```
# Get the baseline and monitoring statistics & violations
baseline_statistics = baseline_job.baseline_statistics().body_dict
execution_statistics = execution.statistics().body_dict
violations = execution.constraint_violations().body_dict['violations']
```

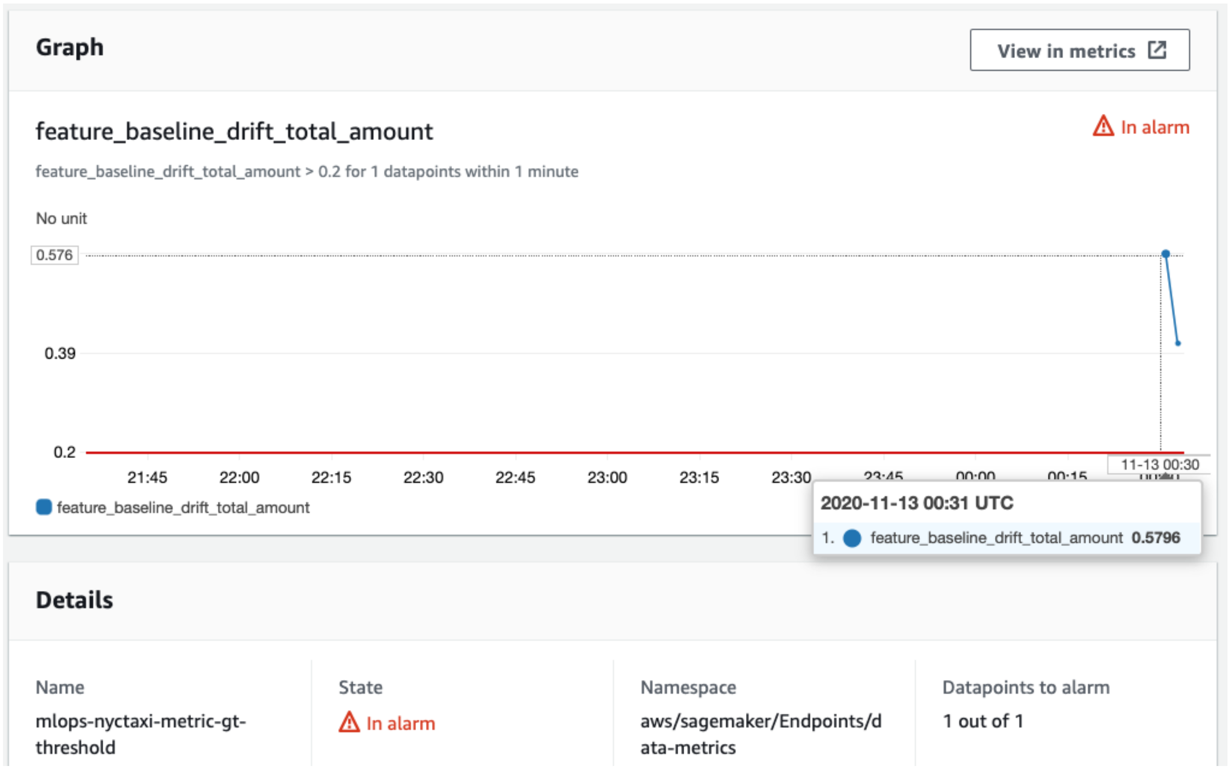
In [ ]:
```
mu.show_violation_df(baseline_statistics=baseline_statistics,
                     latest_statistics=execution_statistics,
                     violations=violations)
```

Trigger Retraining

The CodePipeline instance is configured with [CloudWatch Events (https://docs.aws.amazon.com/codepipeline/latest/userguide/create-cloudtrail-S3-source.html)](https://docs.aws.amazon.com/codepipeline/latest/userguide/create-cloudtrail-S3-source.html) to start the pipeline for retraining when the drift detection triggers specific metric alarms.

You can simulate drift by putting a metric value above the threshold of `0.2` directly into CloudWatch. This will trigger the alarm, and start the code pipeline.

Tip: This alarm is configured only for the latest production endpoint, so re-training will only occur if you are putting metrics against the latest endpoint.


Graph
View in metrics
feature_baseline_drift_total_amount
In alarm
feature_baseline_drift_total_amount > 0.2 for 1 datapoints within 1 minute
No unit
0.576
0.39
0.2
21:45
22:00
22:15
22:30
22:45
23:00
23:15
23:30
23:45
00:00
00:15
11-13 00:30
feature_baseline_drift_total_amount
2020-11-13 00:31 UTC
1. feature_baseline_drift_total_amount 0.5796
Details

Name	State	Namespace	Datapoints to alarm
mlops-nyctaxi-metric-gt-threshold	In alarm	aws/sagemaker/Endpoints/data-metrics	1 out of 1

Run the code below to trigger the metric alarm. The cell output will be a link to CloudWatch, where you can see the alarm (similar to the screenshot above), and a link to CodePipeline which you will see run again. Note that it can take a couple of minutes for everything to trigger.

In [ ]:
```
from datetime import datetime
import random

cloudwatch = boto3.client('cloudwatch')

# Define the metric name and threshold
metric_name = 'feature_baseline_drift_total_amount'
metric_threshold = 0.2

# Put a new metric to trigger an alaram
def put_drift_metric(value):
    print('Putting metric: {}'.format(value))
    response = cloudwatch.put_metric_data(
        Namespace='aws/sagemaker/Endpoints/data-metrics',
        MetricData=[
            {
                'MetricName': metric_name,
                'Dimensions': [
                    {
                        'Name': 'MonitoringSchedule',
                        'Value': schedule_name
                    },
                    {
                        'Name': 'Endpoint',
                        'Value': prd_endpoint_name
                    },
                ],
                'Timestamp': datetime.now(),
                'Value': value,
                'Unit': 'None'
            },
        ]
    )

def get_drift_stats():
    response = cloudwatch.get_metric_statistics(
        Namespace='aws/sagemaker/Endpoints/data-metrics',
        MetricName=metric_name,
        Dimensions=[
            {
                'Name': 'MonitoringSchedule',
                'Value': schedule_name
            },
            {
                'Name': 'Endpoint',
                'Value': prd_endpoint_name
            },
        ],
        StartTime=datetime.now() - timedelta(minutes=2),
        EndTime=datetime.now(),
        Period=1,
        Statistics=['Average'],
        Unit='None'
    )
    if 'Datapoints' in response and len(response['Datapoints']) > 0:
        return response['Datapoints'][0]['Average']
    return 0

print('Simluate drift on endpoint: {}'.format(prd_endpoint_name))

while True:
    put_drift_metric(round(random.uniform(metric_threshold, 1.0), 4))
    drift_stats = get_drift_stats()
    print('Average drift amount: {}'.format(get_drift_stats()))
    if drift_stats > metric_threshold:
        break
    time.sleep(1)
```

Click through to the Alarm and CodePipeline Execution history with the links below.

In [ ]:
```
# Output a html link to the cloudwatch dashboard
metric_alarm_name = 'mlops-{}-metric-gt-threshold'.format(model_name)
HTML('''<a target="_blank" href="https://{0}.console.aws.amazon.com/cloudwatch/h
     <a target="_blank" href="https://{0}.console.aws.amazon.com/codesuite/codep
```

Once the pipeline is running again you can jump back up to Inspect Training Job

Create Synthetic Monitoring

Amazon CloudWatch Synthetics (https://aws.amazon.com/blogs/aws/new-use-cloudwatch-synthetics-to-monitor-sites-api-endpoints-web-workflows-and-more/) allows you to monitor sites, REST APIs, and other services deployed on AWS. You can set up a canary to test that your REST API is returning an expected value at a regular interval. This is a great way to validate that the blue/green deployment is not causing any downtime for your end-users.

Use the code below to set up a canary to continuously test the production deployment. This canary simply pings the REST API to test if it is live, using code from `notebook/canary.js` .

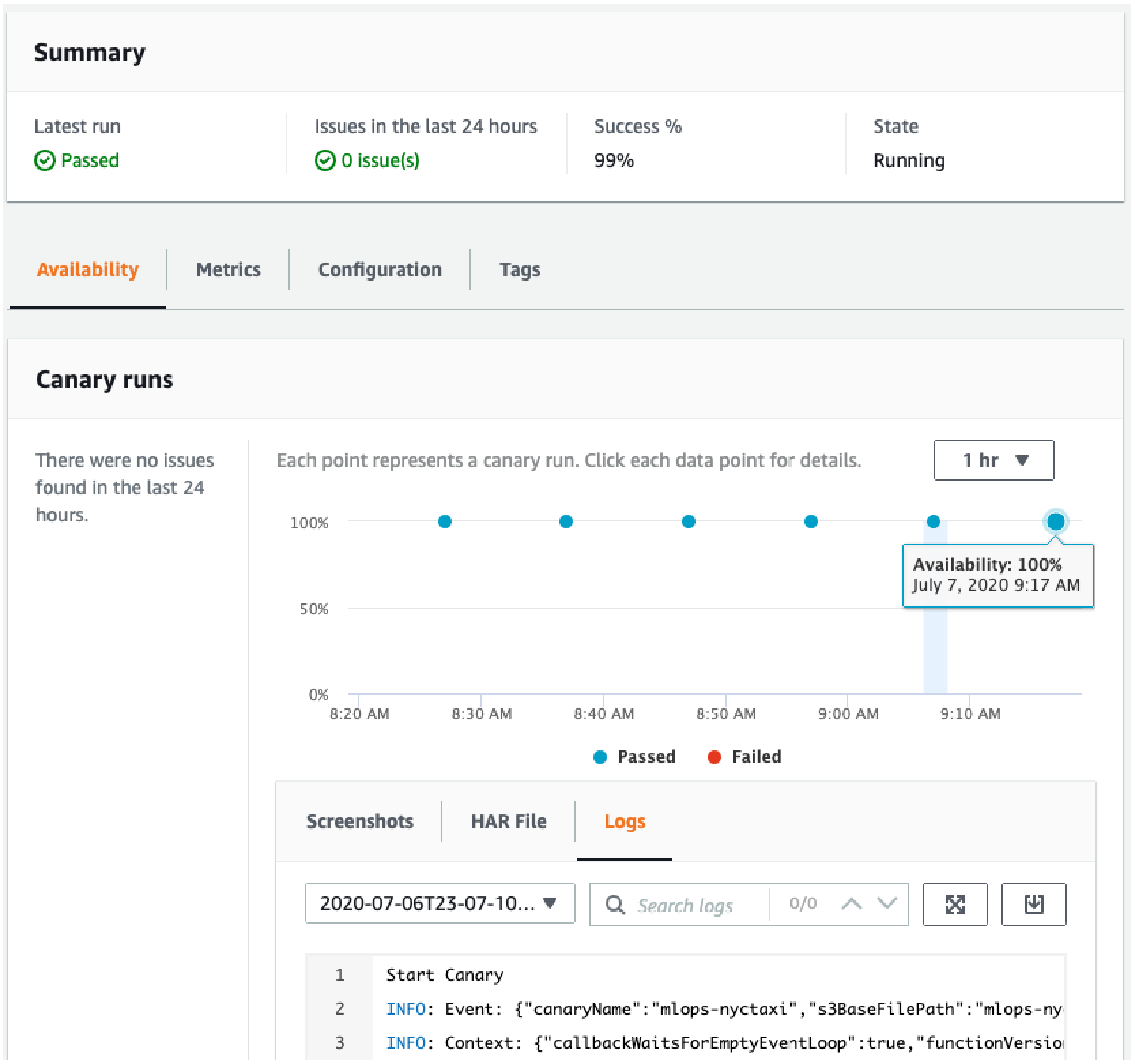
In [ ]:
```python
from urllib.parse import urlparse
from string import Template
from io import BytesIO
import zipfile

# Format the canary_js with rest_api and payload
rest_url = urlparse(rest_api)

with open('canary.js') as f:
    canary_js = Template(f.read()).substitute(hostname=rest_url.netloc, path=res
                                              data=payload.decode('utf-8').strip
# Write the zip file
zip_buffer = BytesIO()
with zipfile.ZipFile(zip_buffer, 'w') as zf:
    zip_path = 'nodejs/node_modules/apiCanaryBlueprint.js' # Set a valid path
    zip_info = zipfile.ZipInfo(zip_path)
    zip_info.external_attr = 0o0755 << 16 # Ensure the file is readable
    zf.writestr(zip_info, canary_js)
zip_buffer.seek(0)

# Create the canary
synth = boto3.client('synthetics')

role = sagemaker.get_execution_role()
s3_canary_uri = 's3://{}/{}'.format(artifact_bucket, model_name)
canary_name = 'mlops-{}'.format(model_name)

try:
    response = synth.create_canary(
        Name=canary_name,
        Code={
            'ZipFile': bytearray(zip_buffer.read()),
            'Handler': 'apiCanaryBlueprint.handler'
        },
        ArtifactS3Location=s3_canary_uri,
        ExecutionRoleArn=role,
        Schedule={
            'Expression': 'rate(10 minutes)',
            'DurationInSeconds': 0 },
        RunConfig={
            'TimeoutInSeconds': 60,
            'MemoryInMB': 960
        },
        SuccessRetentionPeriodInDays=31,
        FailureRetentionPeriodInDays=31,
        RuntimeVersion='syn-nodejs-2.0',
    )
    print('Creating canary: {}'.format(canary_name))
except ClientError as e:
    if e.response["Error"]["Code"] == "AccessDeniedException":
        print('Canary not supported.') # Not supported in event engine
    else:
        raise(e)
```

Now create a CloudWatch alarm which will trigger if the success rate of the canary drops below

90%.

```
In [ ]: cloudwatch = boto3.client('cloudwatch')

canary_alarm_name = '{}-synth-lt-threshold'.format(canary_name)

response = cloudwatch.put_metric_alarm(
    AlarmName=canary_alarm_name,
    ComparisonOperator='LessThanThreshold',
    EvaluationPeriods=1,
    DatapointsToAlarm=1,
    Period=600, # 10 minute interval
    Statistic='Average',
    Threshold=90.0,
    ActionsEnabled=False,
    AlarmDescription='SuccessPercent LessThanThreshold 90%',
    Namespace='CloudWatchSynthetics',
    MetricName='SuccessPercent',
    Dimensions=[
        {
          'Name': 'CanaryName',
          'Value': canary_name
        },
    ],
    Unit='Seconds'
)

print('Creating alarm: {}'.format(canary_alarm_name))
```

Run the code below to check if the canary is running succesfully. The cell will output a link to your CloudWatch Canaries UI, where you can watch the results over time (see screenshot). It can take a couple of minutes for the canary to deploy.



Summary

Latest run	Issues in the last 24 hours	Success %	State
✓ Passed	✓ 0 issue(s)	99%	Running

Canary runs

There were no issues found in the last 24 hours.

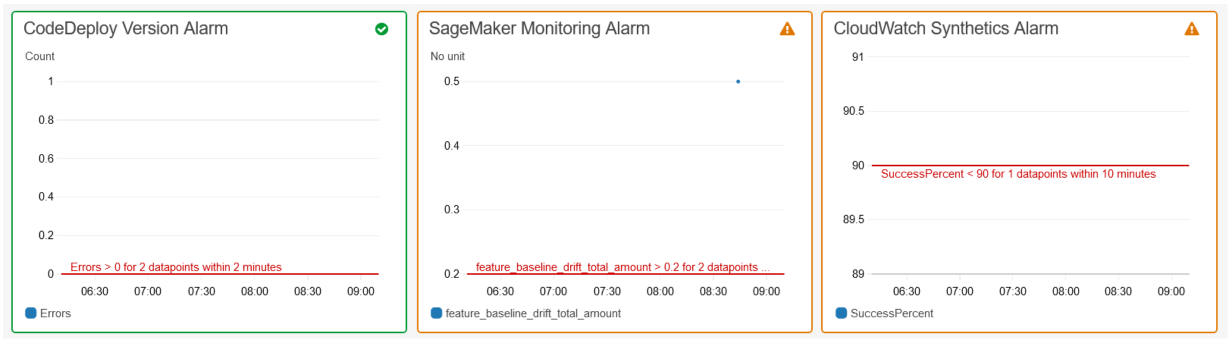
Each point represents a canary run. Click each data point for details.

```
Start Canary
INFO: Event: {"canaryName":"mlops-nyctaxi","s3BaseFilePath":"mlops-ny
INFO: Context: {"callbackWaitsForEmptyEventLoop":true,"functionVersio
```

```
In [ ]: while True:
            try:
                response = synth.get_canary(Name=canary_name)
                status = response['Canary']['Status']['State']
                print('Canary status: {}'.format(status))
                if status == 'ERROR':
                    raise(Exception(response['Canary']['Status']['StateReason']))
                elif status == 'READY':
                    synth.start_canary(Name=canary_name)
                elif status == 'RUNNING':
                    break
            except ClientError as e:
                if e.response["Error"]["Code"] == "ResourceNotFoundException":
                    print('No canary found.')
                    break
                elif e.response["Error"]["Code"] == "AccessDeniedException":
                    print('Canary not supported.') # Not supported in event engine
                    break
                print(e.response["Error"]["Message"])
            time.sleep(10)

        # Output a html link to the cloudwatch console
        HTML('<a target="_blank" href="https://{0}.console.aws.amazon.com/cloudwatch/home
```

Create a CloudWatch dashboard

Finally, use the code below to create a CloudWatch dashboard to visualize the key performance metrics and alarms which you have created during this demo. The cell will output a link to the dashboard. This dashboard shows 9 charts in three rows, where the first row displays Lambda metrics, the second row displays SageMaker metrics, and the third row (shown in the screenshot below) displays the alarms set up for the pipeline.


CodeDeploy Version Alarm
Count
Errors > 0 for 2 datapoints within 2 minutes
Errors
SageMaker Monitoring Alarm
No unit
feature_baseline_drift_total_amount > 0.2 for 2 datapoints ...
feature_baseline_drift_total_amount
CloudWatch Synthetics Alarm
SuccessPercent < 90 for 1 datapoints within 10 minutes
SuccessPercent
06:30 07:00 07:30 08:00 08:30 09:00

In [ ]:
```
sts = boto3.client('sts')
account_id = sts.get_caller_identity().get('Account')
dashboard_name = 'mlops-{}'.format(model_name)

with open('dashboard.json') as f:
    dashboard_body = Template(f.read()).substitute(region=region, account_id=acco
    response = cloudwatch.put_dashboard(
        DashboardName=dashboard_name,
        DashboardBody=dashboard_body
    )

# Output a html link to the cloudwatch dashboard
HTML('<a target="_blank" href="https://{0}.console.aws.amazon.com/cloudwatch/home
```

Congratulations! You have made it to the end of this notebook, and have automated a safe MLOps pipeline using a wide range of AWS services.

You can use the other notebook in this repository workflow.ipynb (workflow.ipynb) to implement your own ML model and deploy it as part of this pipeline. Or, if you are finished with the content, follow the instructions in the next section to clean up the resources you have deployed.

Cleanup

Execute the following cell to delete the stacks created in the pipeline. For a model name of **nyctaxi** these would be:

1. *nyctaxi*-deploy-prd
2. *nyctaxi*-deploy-dev
3. *nyctaxi*-workflow
4. sagemaker-custom-resource

In [ ]:
```
cfn = boto3.client('cloudformation')

# Delete the prod and then dev stack
for stack_name in [f'{pipeline_name}-deploy-prd',
                   f'{pipeline_name}-deploy-dev',
                   f'{pipeline_name}-workflow',
                   'sagemaker-custom-resource']:
    print('Deleting stack: {}'.format(stack_name))
    cfn.delete_stack(StackName=stack_name)
    cfn.get_waiter('stack_delete_complete').wait(StackName=stack_name)
```

The following code will stop and delete the canary you created.

In [ ]:
```
while True:
    try:
        response = synth.get_canary(Name=canary_name)
        status = response['Canary']['Status']['State']
        print('Canary status: {}'.format(status))
        if status == 'ERROR':
            raise(Exception(response['Canary']['Status']['StateReason']))
        elif status == 'STOPPED':
            synth.delete_canary(Name=canary_name)
        elif status == 'RUNNING':
            synth.stop_canary(Name=canary_name)
    except ClientError as e:
        if e.response["Error"]["Code"] == "ResourceNotFoundException":
            print('Canary succesfully deleted.')
            break
        elif e.response["Error"]["Code"] == "AccessDeniedException":
            print('Canary not created.') # Not supported in event engine
            break
        print(e.response["Error"]["Message"])
    time.sleep(10)
```

The following code will delete the dashboard.

In [ ]:
```
cloudwatch.delete_alarms(AlarmNames=[canary_alarm_name])
print('Alarm deleted')

cloudwatch.delete_dashboards(DashboardNames=[dashboard_name])
print('Dashboard deleted')
```

Finally, close this notebook and you can delete the CloudFormation you created to launch this MLOps sample.