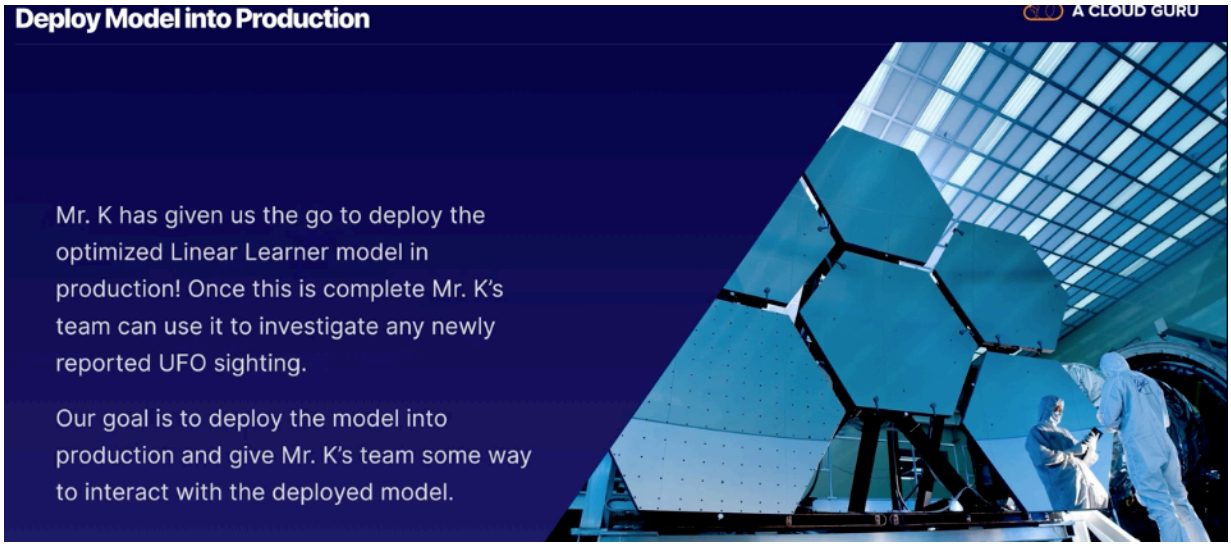
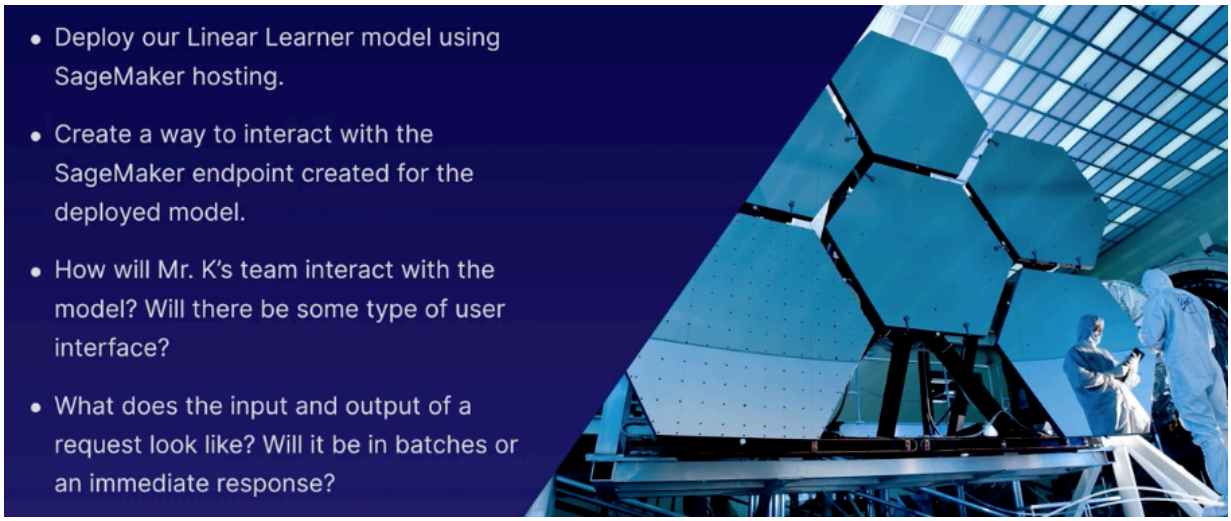
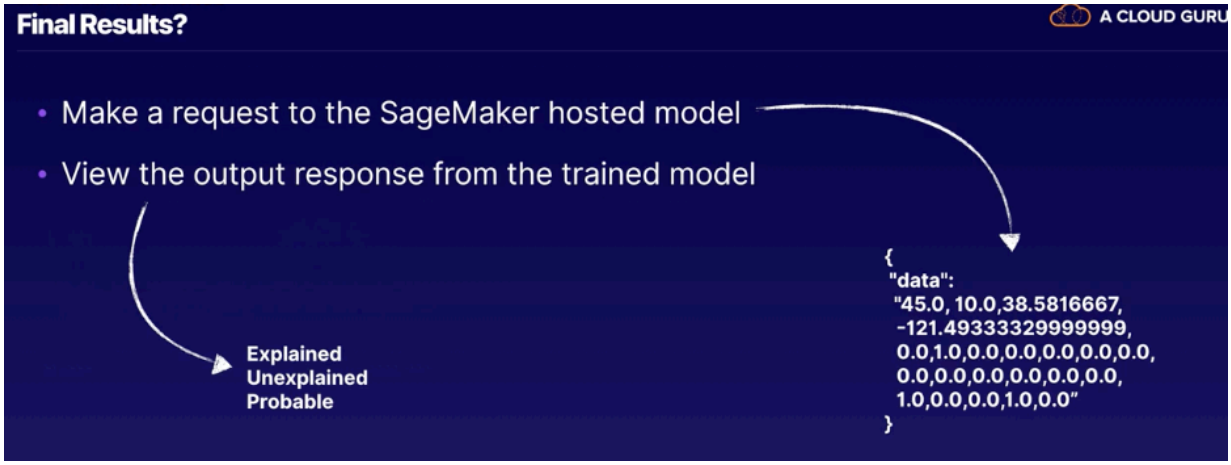
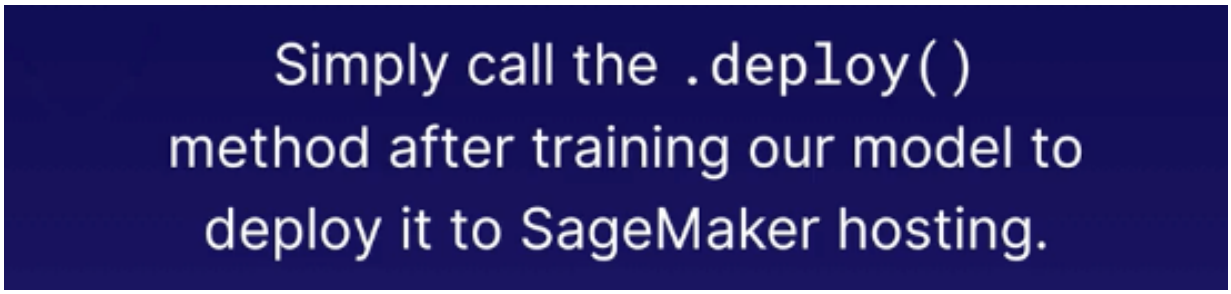
Cloud Guru - 8 - Implementation and Operations Lab



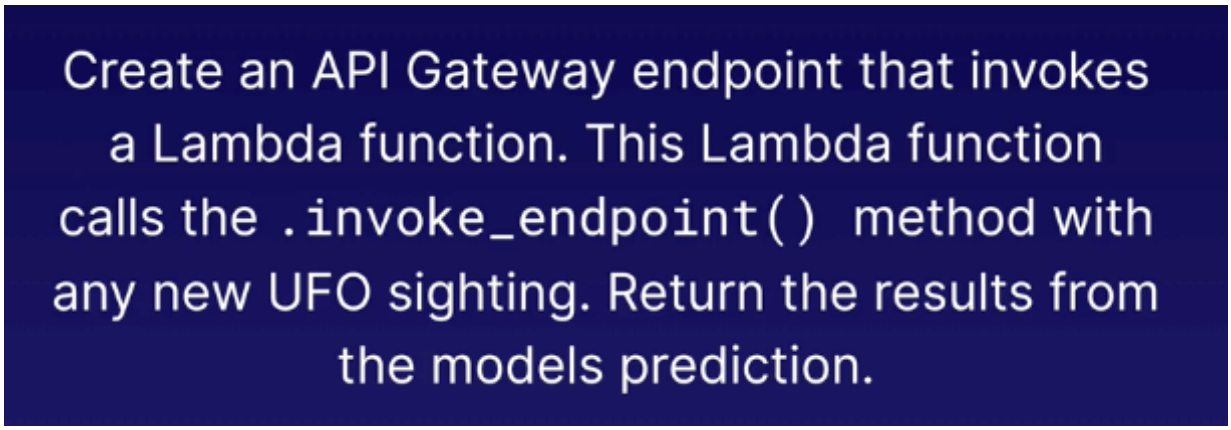


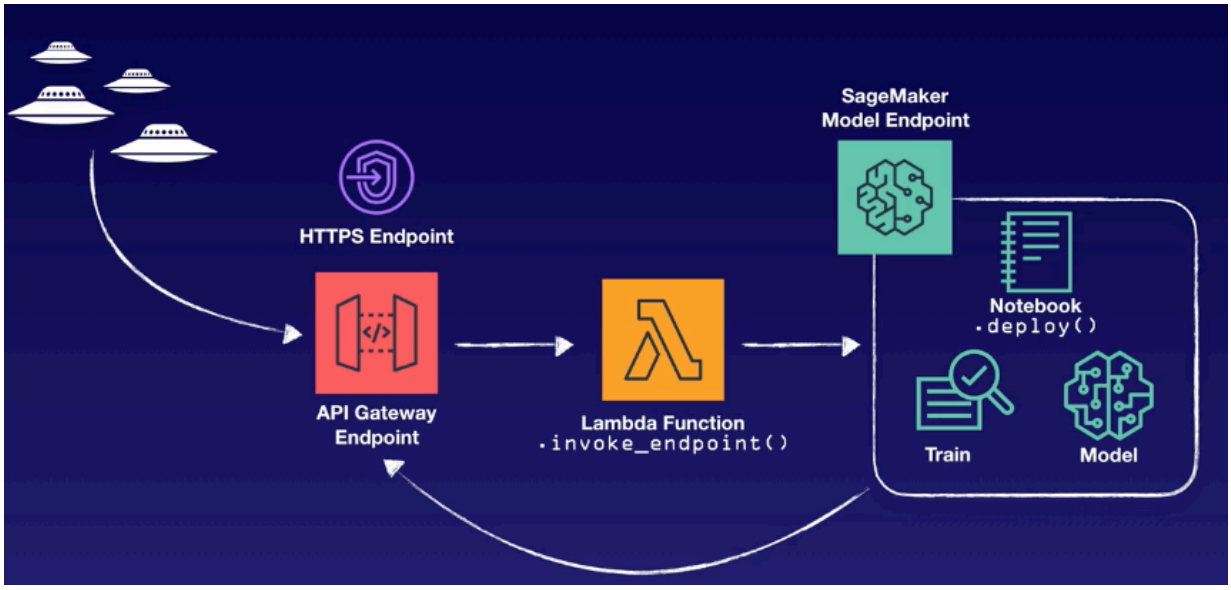




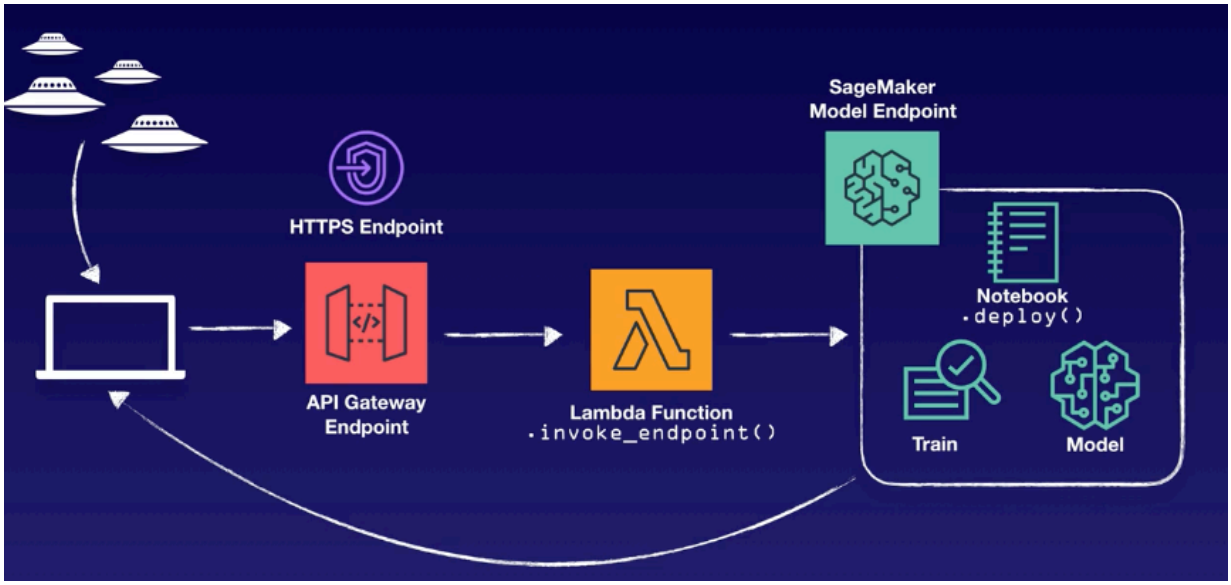
=> just one line of code in Jupiter

After deployment:



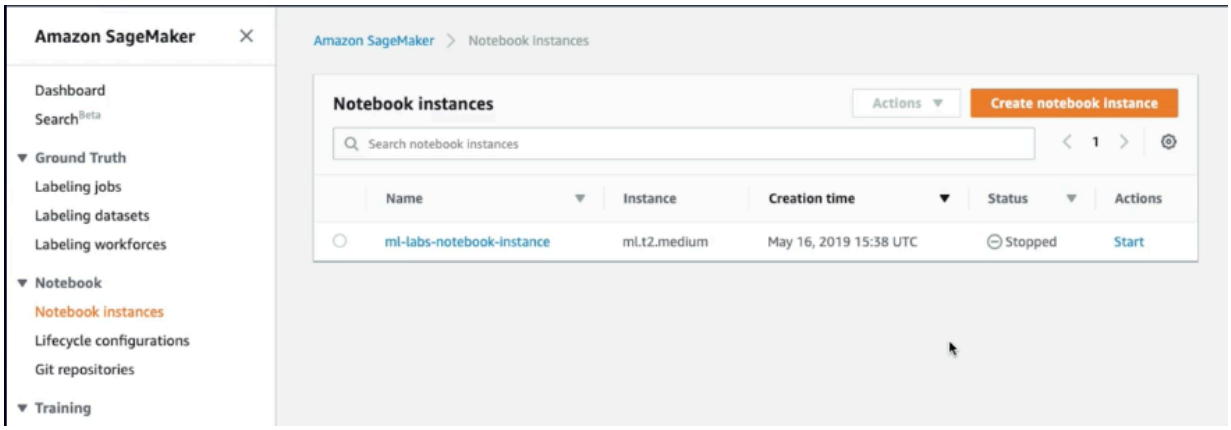


To note we can create an application fronting the API endpoint as a mobile or web app - but that is a bit out of scope for this lab

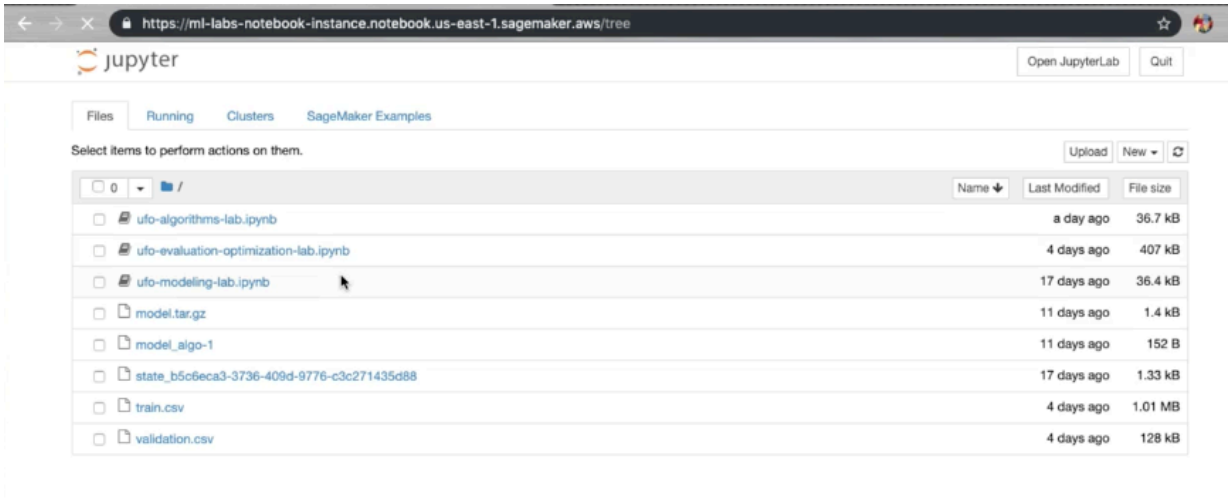


Hands-on Lab

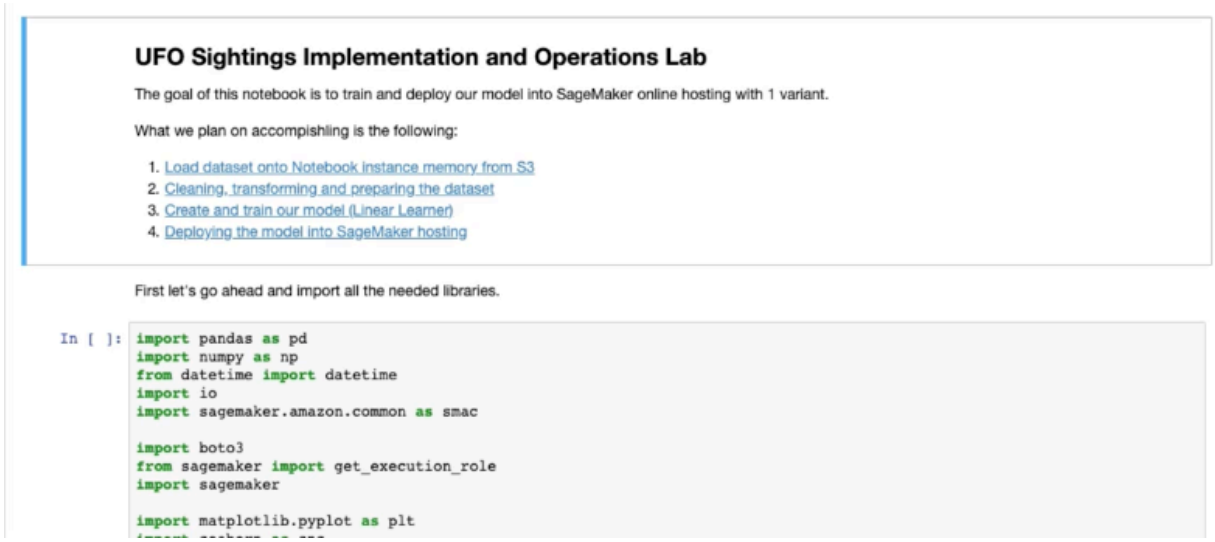
Sagemaker - > Start notebook from last lab



Open Jupiter notebook, upload notebook from resource section of the lab



Upload notebook and open it



Load dataset from S3 into memory into a data frame

Let's get the UFO sightings data that is stored in S3 and load it into memory.

In [2]:
```
role = get_execution_role()
bucket='ml-labs-acg'
sub_folder = 'ufo_dataset'
data_key = 'ufo_fullset.csv'
data_location = 's3://{}/{}/{}'.format(bucket, sub_folder, data_key)

df = pd.read_csv(data_location, low_memory=False)
df.head()
```

Out[2]:

	reportedTimestamp	eventDate	eventTime	shape	duration	witnesses	weather	firstName	lastName	latitude	longitude	sighting	physicalEvidence
0	1977-04-04T04:02:23.340Z	1977-03-31	23:46	circle	4	1	rain	Ila	Bashirian	47.329444	-122.578889	Y	N
1	1982-11-22T02:06:32.019Z	1982-11-15	22:04	disk	4	1	partly cloudy	Eriberto	Runolfsson	52.664913	-1.034894	Y	Y
2	1992-12-07T19:06:52.482Z	1992-12-07	19:01	circle	49	1	clear	Miller	Watsica	38.951667	-92.333889	Y	N
3	2011-02-24T21:06:34.898Z	2011-02-21	20:56	disk	13	1	partly cloudy	Clifton	Bechtelar	41.496944	-71.367778	Y	N
4	1991-03-09T16:18:45.501Z	1991-03-09	11:42	circle	17	1	mostly cloudy	Jayda	Ebert	47.606389	-122.330833	Y	N

Step 2: Cleaning, transforming and preparing the dataset

This step is so important. It's crucial that we clean and prepare our data before we do anything else.

Let's go ahead and start preparing our dataset by transforming some of the values into the correct data types. Here is what we are going to take care of.

1. Convert the `reportedTimestamp` and `eventDate` to a datetime data types.
2. Convert the `shape` and `weather` to a category data type.
3. Map the `physicalEvidence` and `contact` from 'Y', 'N' to `0`, `1`.
4. Convert the `researchOutcome` to a category data type (target attribute).

Let's also drop the columns that are not important.

1. We can drop `sighting` becuase it is always 'Y' or Yes.
2. Let's drop the `firstName` and `lastName` becuase they are not important in determining the `researchOutcome`.
3. Let's drop the `reportedTimestamp` becuase when the sighting was reporting isn't going to help us determine the legitimacy of the sighting.
4. We would need to create some sort of buckets for the `eventDate` and `eventTime`, like seasons for example, but since the distribution of dates is pretty even, let's go ahead and drop them.

Finally, let's apply one-hot encoding

1. We need to one-hot both the `weather` attribute and the `shape` attribute.
2. We also need to transform or map the researchOutcome (target) attribute into numeric values. This is what the alogrithm is expecting. We can do this by mapping unexplained, explained, and probable to 0, 1, 2.

In [ ]:
```
# Replace the missing values with the most common shape
df['shape'] = df['shape'].fillna(df['shape'].value_counts().index[0])

df['reportedTimestamp'] = pd.to_datetime(df['reportedTimestamp'])
df['eventDate'] = pd.to_datetime(df['eventDate'])

df['shape'] = df['shape'].astype('category')
df['weather'] = df['weather'].astype('category')

df['physicalEvidence'] = df['physicalEvidence'].replace({'Y': 1, 'N': 0})
df['contact'] = df['contact'].replace({'Y': 1, 'N': 0})

df['researchOutcome'] = df['researchOutcome'].astype('category')

df.drop(columns=['firstName', 'lastName', 'sighting', 'reportedTimestamp', 'eventDate', 'eventTime'], inplace=True)

# Let's one-hot the weather and shape attribute
df = pd.get_dummies(df, columns=['weather', 'shape'])

# Let's replace the researchOutcome values with 0, 1, 2 for Unexplained, Explained, and Probable
df['researchOutcome'] = df['researchOutcome'].replace({'unexplained': 0, 'explained': 1, 'probable': 2})
```

In [ ]:
```
display(df.head())
display(df.shape)
```

In [4]:
```
display(df.head())
display(df.shape)
```

	duration	witnesses	latitude	longitude	physicalEvidence	contact	researchOutcome	weather_clear	weather_fog	weather_mostly cloudy	...	weather_stormy
0	4	1	47.329444	-122.578889	0	0	1	0	0	0	...	0
1	4	1	52.664913	-1.034894	1	0	1	0	0	0	...	0
2	49	1	38.951667	-92.333889	0	0	1	1	0	0	...	0
3	13	1	41.496944	-71.367778	0	0	1	0	0	0	...	0
4	17	1	47.606389	-122.330833	0	0	1	0	0	1	...	0

5 rows × 23 columns

```
(18000, 23)
```

Step 3: Creating and training our model (Linear Learner)

Let's evaluate the Linear Learner algorithm as well. Let's go ahead and randomize the data again and get it ready for the Linear Leaner algorithm. We will also rearrange the columns so it is ready for the algorithm (it expects the first column to be the target attribute)

In [ ]:
```
np.random.seed(0)
rand_split = np.random.rand(len(df))
train_list = rand_split < 0.8
val_list = (rand_split >= 0.8) & (rand_split < 0.9)
test_list = rand_split >= 0.9

 # This dataset will be used to train the model.
data_train = df[train_list]

# This dataset will be used to validate the model.
data_val = df[val_list]

# This dataset will be used to test the model.
data_test = df[test_list]

# Breaks the datasets into attribute numpy.ndarray and the same for target attribute.
train_X = data_train.drop(columns='researchOutcome').as_matrix()
train_y = data_train['researchOutcome'].as_matrix()

val_X = data_val.drop(columns='researchOutcome').as_matrix()
val_y = data_val['researchOutcome'].as_matrix()

test_X = data_test.drop(columns='researchOutcome').as_matrix()
test_y = data_test['researchOutcome'].as_matrix()
```

In [7]:
```
train_file = 'ufo_sightings_train_recordIO_protobuf.data'

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, train_X.astype('float32'), train_y.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object('implementation_operations_lab/linearlearner_train/{}'.format(trai
training_recordIO_protobuf_location = 's3://{}/implementation_operations_lab/linearlearner_train/{}'.format(bucket, tra
```

```
The Pipe mode recordIO protobuf training data: s3://ml-labs-acg/implementation_operations_lab/linearlearner_train/ufo
_sightings_train_recordIO_protobuf.data
```

Let's create recordIO file for the validation data and upload it to S3

In [8]:
```
validation_file = 'ufo_sightings_validatioin_recordIO_protobuf.data'

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, val_X.astype('float32'), val_y.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object('implementation_operations_lab/linearlearner_validation/{}'.format
validate_recordIO_protobuf_location = 's3://{}/implementation_operations_lab/linearlearner_validation/{}'.format(bucket
print('The Pipe mode recordIO protobuf validation data: {}'.format(validate_recordIO_protobuf_location))
```

```
The Pipe mode recordIO protobuf validation data: s3://ml-labs-acg/implementation_operations_lab/linearlearner_validat
ion/ufo_sightings_validatioin_recordIO_protobuf.data
```

In [9]:
```
from sagemaker.amazon.amazon_estimator import get_image_uri
import sagemaker

container = get_image_uri(boto3.Session().region_name, 'linear-learner', "1")
```

In [10]:
```
# Create a training job name
job_name = 'ufo-linear-learner-job-{}'.format(datetime.now().strftime("%Y%m%d%H%M%S"))
print('Here is the job name {}'.format(job_name))

# Here is where the model-artifact will be stored
output_location = 's3://{}/implementation_operations_lab/linearlearner_output'.format(bucket)
```

```
Here is the job name ufo-linear-learner-job-20190603200343
```

Next we start building out our model by using the SageMaker Python SDK and passing in everything that is required to create a Linear Learner model.

First I like to always create a specific job name. Next, we'll need to specify training parameters.

Finally, after everything is included and ready, then we can call the `.fit()` function which specifies the S3 location for training and validation data.

In [ ]:
```
print('The feature_dim hyperparameter needs to be set to {}.'.format(data_train.shape[1] - 1))
```

In [11]:
```
print('The feature_dim hyperparameter needs to be set to {}.'.format(data_train.shape[1] - 1))
```

```
The feature_dim hyperparameter needs to be set to 22.
```

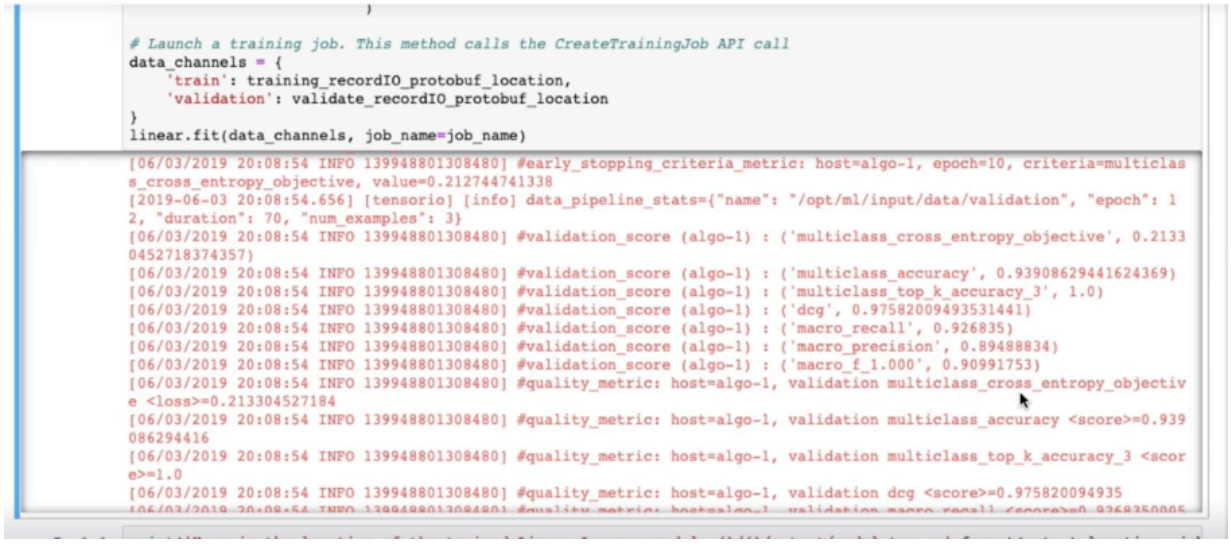
In [ ]:
```
sess = sagemaker.Session()

# Setup the LinearLeaner algorithm from the ECR container
linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count=1,
                                       train_instance_type='ml.c4.xlarge',
                                       output_path=output_location,
                                       sagemaker_session=sess,
                                       input_mode='Pipe')
# Setup the hyperparameters
linear.set_hyperparameters(feature_dim=22,
                           predictor_type='multiclass_classifier',
                           num_classes=3
                            # add optimized hyperparmeters here
                            # add optimized hyperparmeters here
                            # add optimized hyperparmeters here
                            # add optimized hyperparmeters here
                            # add optimized hyperparmeters here
                            # add optimized hyperparmeters here
                          )

# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {
    'train': training_recordIO_protobuf_location,
    'validation': validate_recordIO_protobuf_location
```

```
# Setup the LinearLeaner algorithm from the ECR container
linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count=1,
                                       train_instance_type='ml.c4.xlarge',
                                       output_path=output_location,
                                       sagemaker_session=sess,
                                       input_mode='Pipe')
# Setup the hyperparameters
linear.set_hyperparameters(feature_dim=22,
                           predictor_type='multiclass_classifier',
                           num_classes=3
                           early_stopping_patience=3,
                            early_stopping_tolerance=0.001,
                            epochs=15,
                            l1=0.0647741539306635,
                            learning_rate=0.09329042024421902,
                            loss='auto',
                            mini_batch_size=744,
                            normalize_data='true',
                            normalize_label='auto',
                            num_models='auto',
                            optimizer='auto',
                            unbias_data='auto',
                            unbias_label='auto',
                            use_bias='true',
                            wd=0.000212481391205101
                          )
```

```
# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {
    'train': training_recordIO_protobuf_location,
    'validation': validate_recordIO_protobuf_location
}
linear.fit(data_channels, job_name=job_name)
```

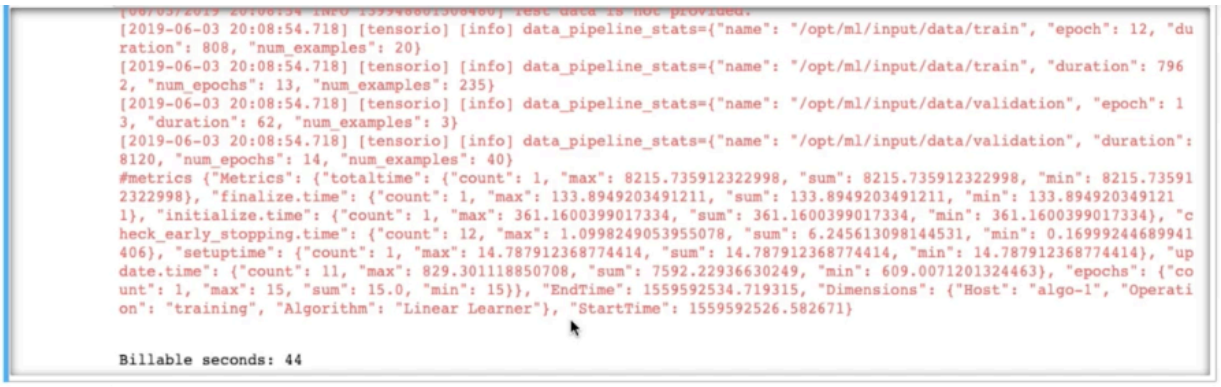
In [ ]:
```
print('Here is the location of the trained Linear Learner model: {}/{}/output/model.tar.gz'.format(output_location, job
```

From here we have our trained model we can deploy into production!

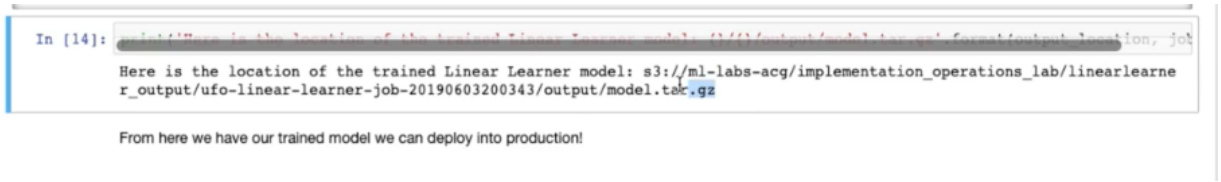


```
# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {
    'train': training_recordIO_protobuf_location,
    'validation': validate_recordIO_protobuf_location
}
linear.fit(data_channels, job_name=job_name)
```

```
[06/03/2019 20:08:54 INFO 139948801308480] #early_stopping_criteria_metric: host=algo-1, epoch=10, criteria=multiclass_cross_entropy_objective, value=0.212744741338
[2019-06-03 20:08:54.656] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "epoch": 12, "duration": 70, "num_examples": 3}
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('multiclass_cross_entropy_objective', 0.21330452718374357)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('multiclass_accuracy', 0.93908629441624369)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('multiclass_top_k_accuracy_3', 1.0)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('dcg', 0.97582009493531441)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('macro_recall', 0.926835)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('macro_precision', 0.89488834)
[06/03/2019 20:08:54 INFO 139948801308480] #validation_score (algo-1) : ('macro_f_1.000', 0.90991753)
[06/03/2019 20:08:54 INFO 139948801308480] #quality_metric: host=algo-1, validation multiclass_cross_entropy_objective <loss>=0.213304527184
[06/03/2019 20:08:54 INFO 139948801308480] #quality_metric: host=algo-1, validation multiclass_accuracy <score>=0.939086294416
[06/03/2019 20:08:54 INFO 139948801308480] #quality_metric: host=algo-1, validation multiclass_top_k_accuracy_3 <score>=1.0
[06/03/2019 20:08:54 INFO 139948801308480] #quality_metric: host=algo-1, validation dcg <score>=0.975820094935
```

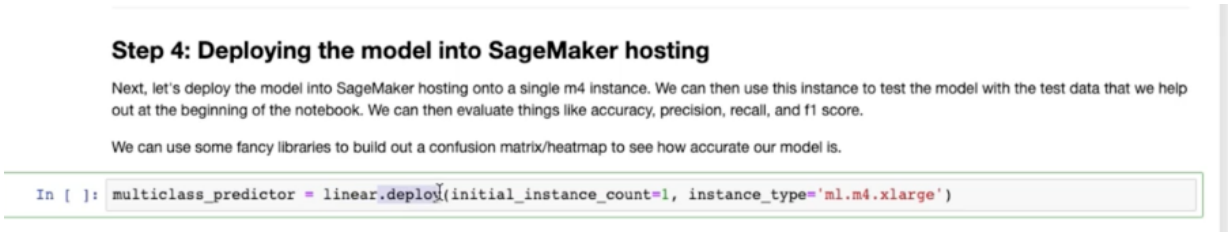


```
[2019-06-03 20:08:54.718] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/train", "epoch": 12, "duration": 808, "num_examples": 20}
[2019-06-03 20:08:54.718] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/train", "duration": 7962, "num_epochs": 13, "num_examples": 235}
[2019-06-03 20:08:54.718] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "epoch": 13, "duration": 62, "num_examples": 3}
[2019-06-03 20:08:54.718] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "duration": 8120, "num_epochs": 14, "num_examples": 40}
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 8215.735912322998, "sum": 8215.735912322998, "min": 8215.735912322998}, "finalize.time": {"count": 1, "max": 133.8949203491211, "sum": 133.8949203491211, "min": 133.8949203491211}, "initialize.time": {"count": 1, "max": 361.1600399017334, "sum": 361.1600399017334, "min": 361.1600399017334}, "check_early_stopping.time": {"count": 12, "max": 1.0998249053955078, "sum": 6.245613098144531, "min": 0.16999244689941406}, "setuptime": {"count": 1, "max": 14.787912368774414, "sum": 14.787912368774414, "min": 14.787912368774414}, "update.time": {"count": 11, "max": 829.301118850708, "sum": 7592.22936630249, "min": 609.0071201324463}, "epochs": {"count": 1, "max": 15, "sum": 15.0, "min": 15}}, "EndTime": 1559592534.719315, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "Linear Learner"}, "StartTime": 1559592526.582671}


Billable seconds: 44
```



```
In [14]: print('Here is the location of the trained Linear Learner model: {}/{}/output/model.tar.gz'.format(output_location, job

Here is the location of the trained Linear Learner model: s3://ml-labs-acg/implementation_operations_lab/linearlearner_output/ufo-linear-learner-job-20190603200343/output/model.tar.gz
```

From here we have our trained model we can deploy into production!



Step 4: Deploying the model into SageMaker hosting

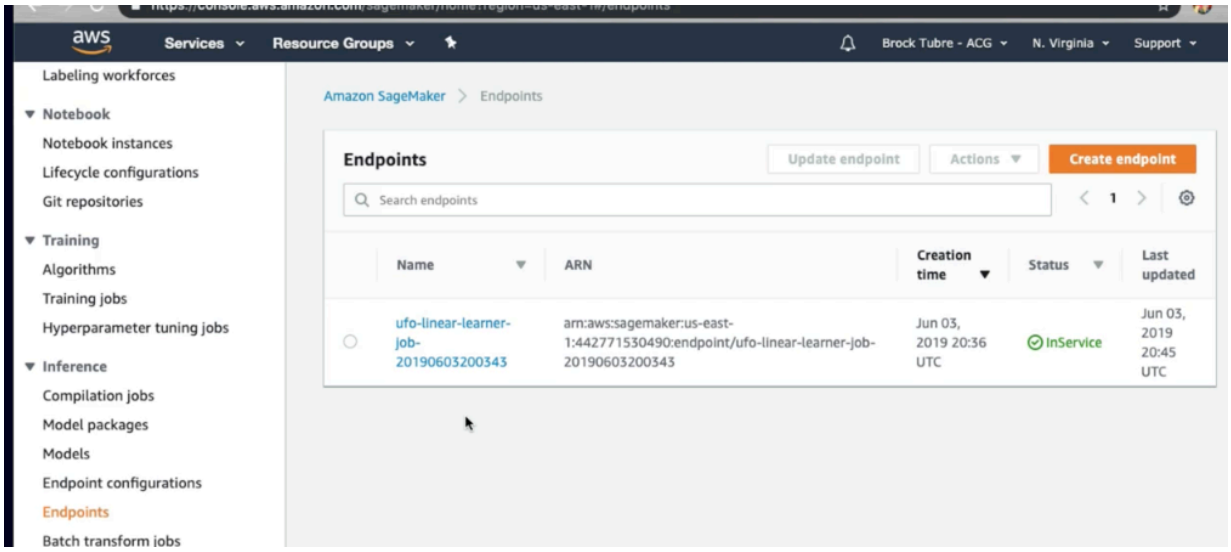
Next, let's deploy the model into SageMaker hosting onto a single m4 instance. We can then use this instance to test the model with the test data that we help out at the beginning of the notebook. We can then evaluate things like accuracy, precision, recall, and f1 score.

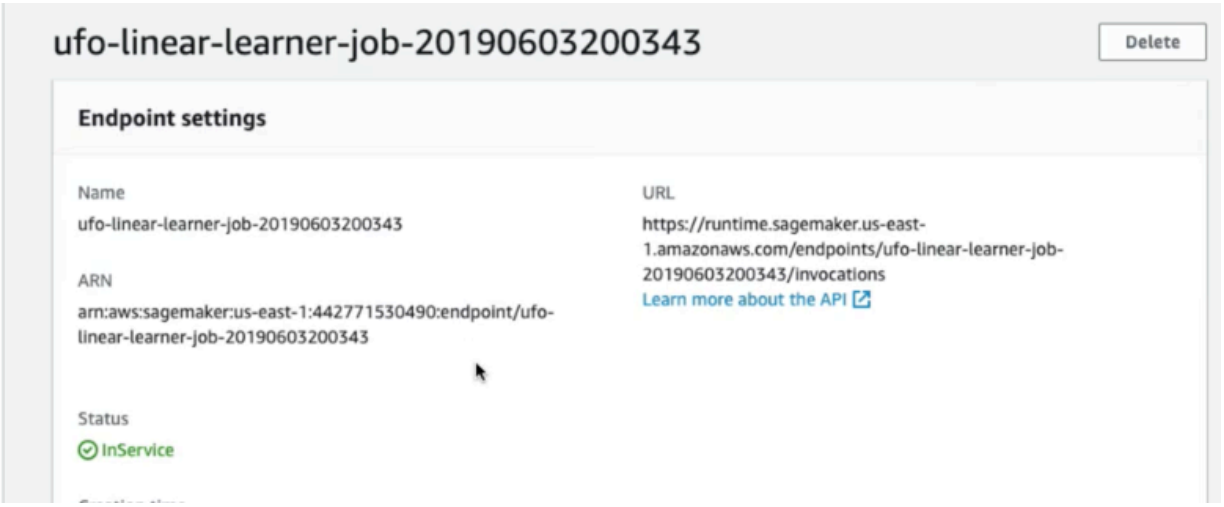
We can use some fancy libraries to build out a confusion matrix/heatmap to see how accurate our model is.

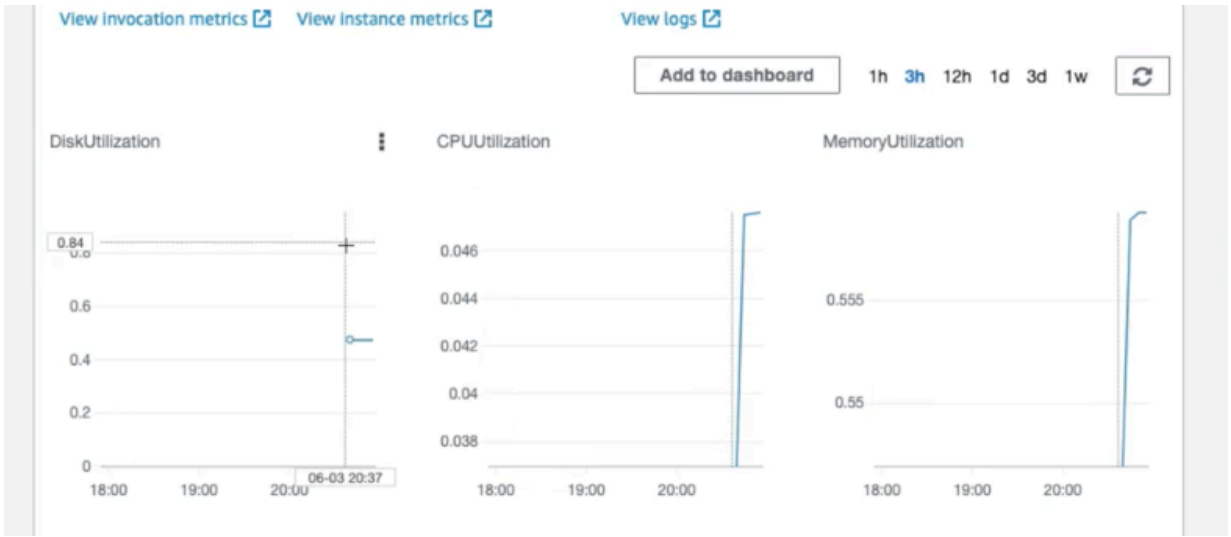
```
In [ ]: multiclass_predictor = linear.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

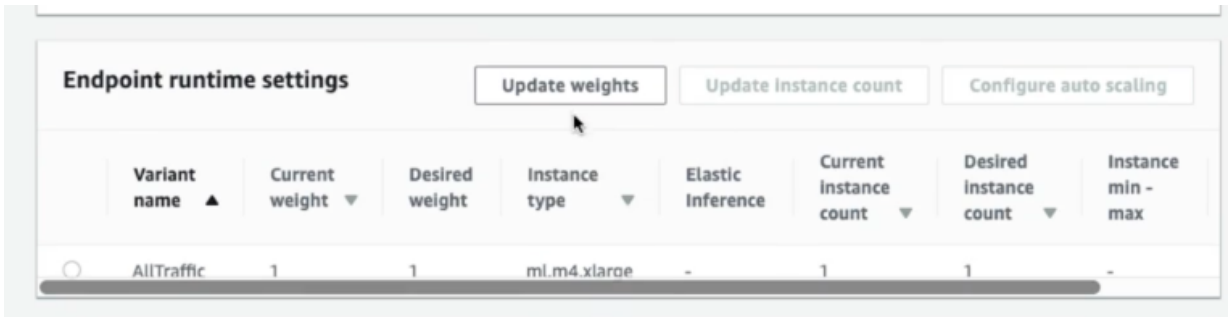
=> deploy() method behind the scene:
- spinning up instance that model will be deployed to: m4.xl (1 instance)
- creating one endpoint on SM hosting
- creating endpoint configuration

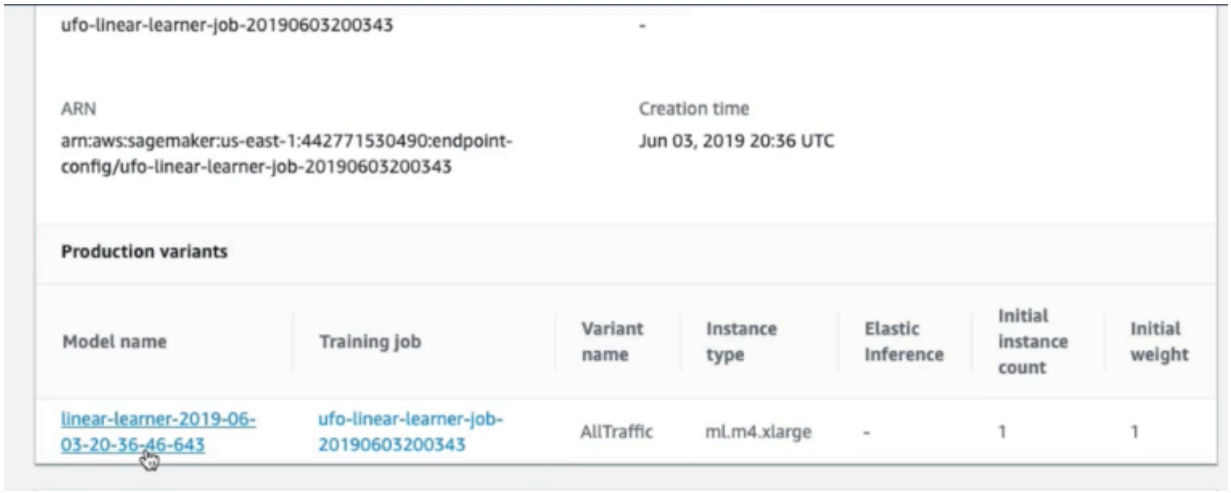
Now we can see an endpoint created and we can use it











This next code is just setup code to allow us to draw out nice and pretty confusion matrix/heatmap.

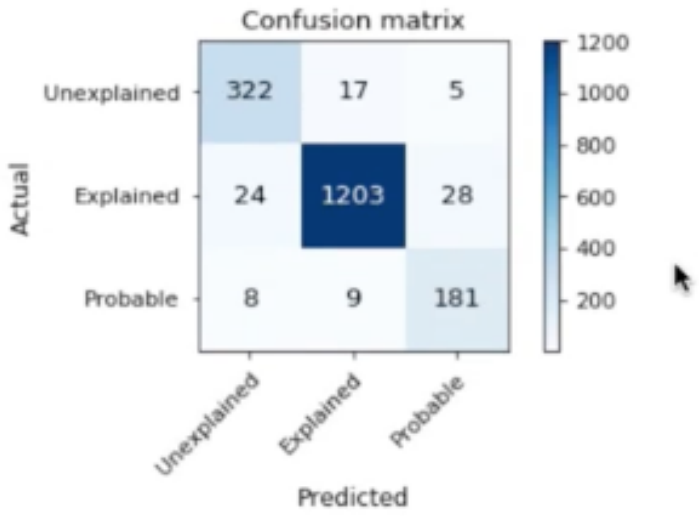
In [ ]:

```
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=None):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
            plt.cm.Greens
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
```

```
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)
```

In [ ]:
```
from sagemaker.predictor import json_deserializer, csv_serializer

multiclass_predictor.content_type = 'text/csv'
multiclass_predictor.serializer = csv_serializer
multiclass_predictor.deserializer = json_deserializer

predictions = []
results = multiclass_predictor.predict(test_X)
predictions += [r['predicted_label'] for r in results['predictions']]
predictions = np.array(predictions)
```

In [ ]:
```
%matplotlib inline
sns.set_context("paper", font_scale=1.4)

y_test = test_y
y_pred = predictions

class_names = np.array(['Unexplained', 'Explained', 'Probable'])

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_test, y_pred, classes=class_names,
                      title='Confusion matrix',
                      cmap=plt.cm.Blues)
plt.grid(False)
plt.show()
```



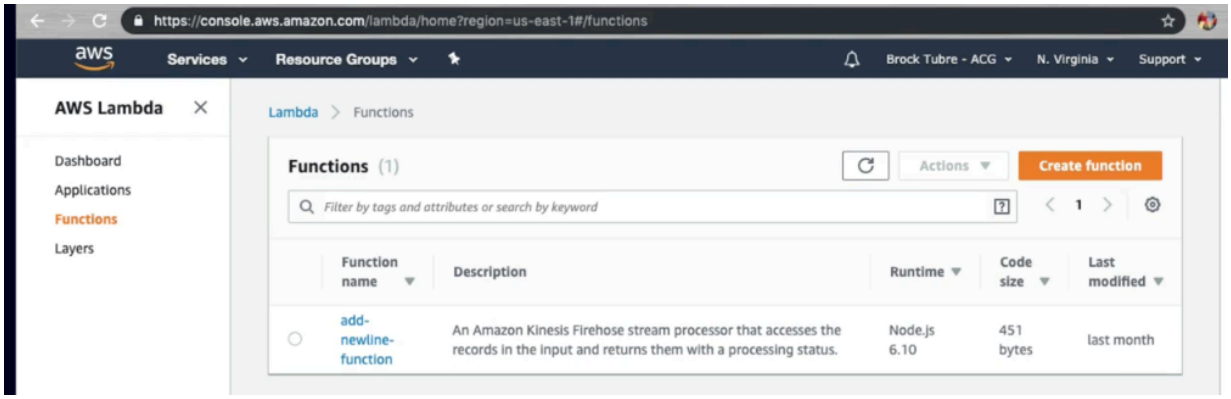
24 Explained but our model predicted unexplained

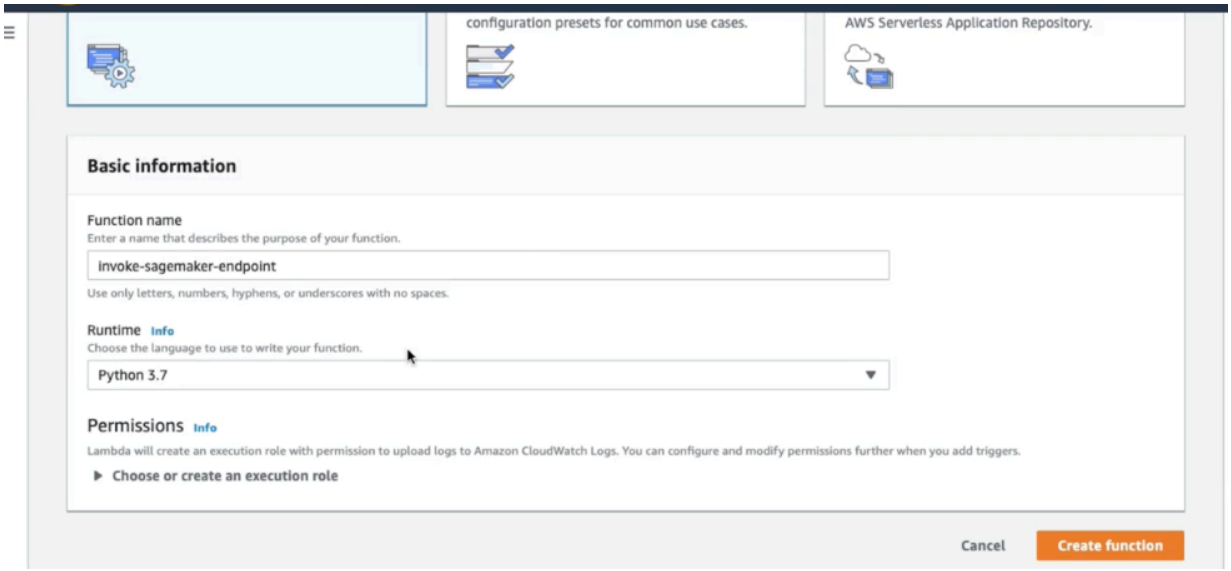
In [19]:
```
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

y_test = data_test['researchOutcome']
y_pred = predictions
scores = precision_recall_fscore_support(y_test, y_pred, average='macro', labels=np.unique(y_pred))
acc = accuracy_score(y_test, y_pred)
print('Accuracy is: {}'.format(acc))
print('Precision is: {}'.format(scores[0]))
print('Recall is: {}'.format(scores[1]))
print('F1 score is: {}'.format(scores[2]))
```

```
Accuracy is: 0.9493600445186422
Precision is: 0.9114145004647342
Recall is: 0.9362512209403713
F1 score is: 0.9232919712224389
```

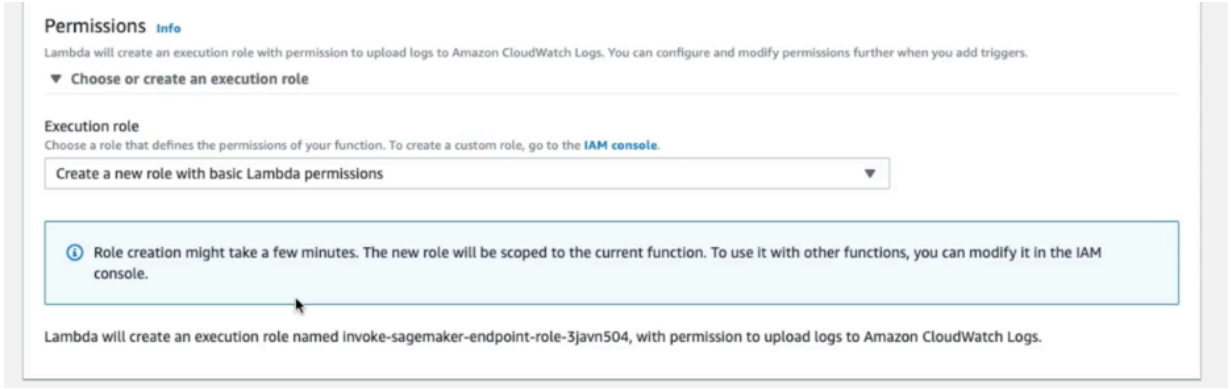
Our model is actually a little more accurate on our test data than on our training data.

Next create a lambda

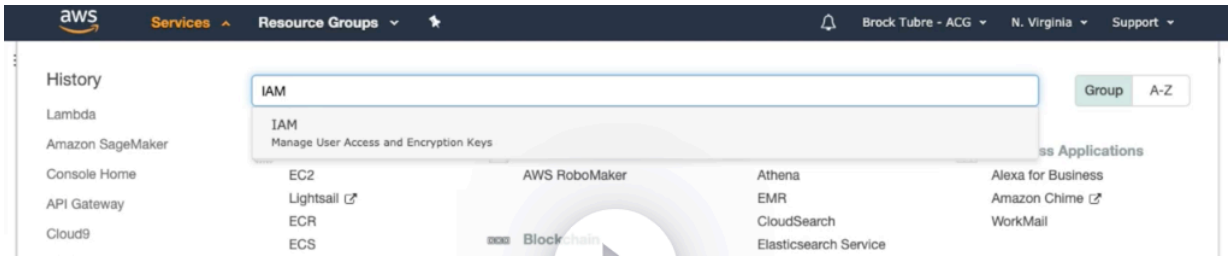




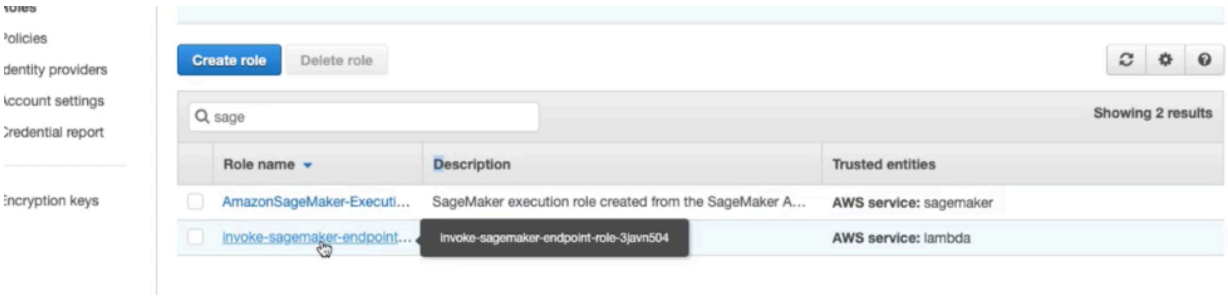
Create an IAM role that will have access to SageMaker



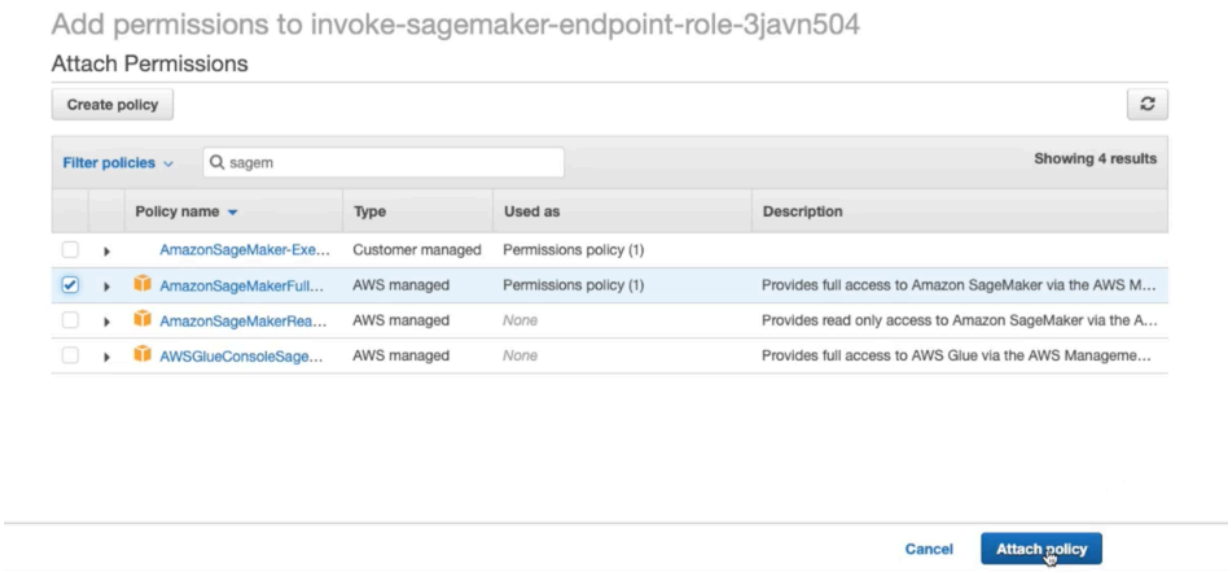
Now go to IAM and give access to SageMaker



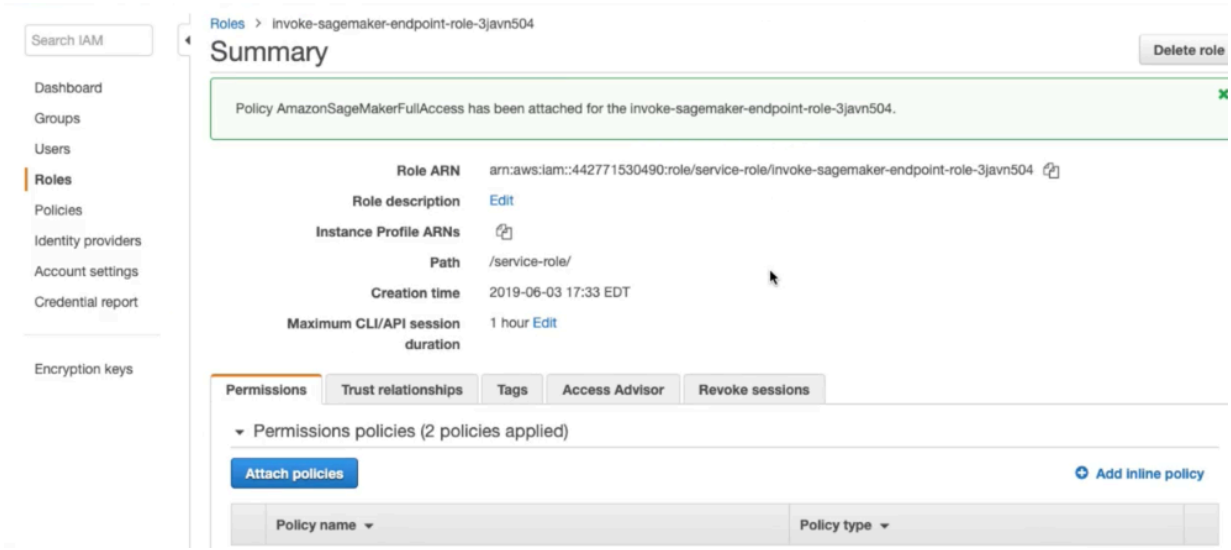
Fine the role we just created:



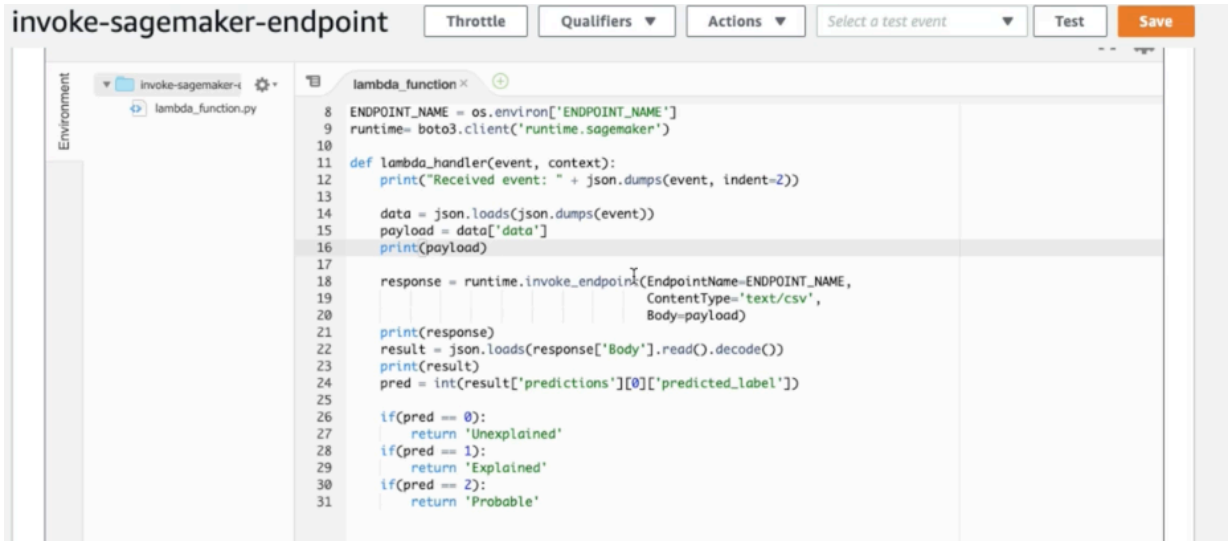
Attach a policy with full access to SageMaker



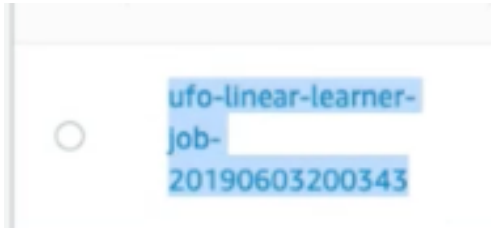
Now we have permission to interact with that pagemaker endpoint

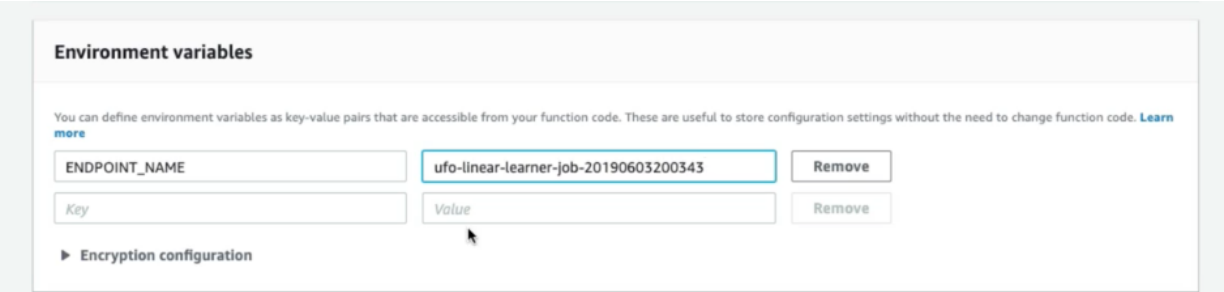


Back to Lambda, insert the code to use

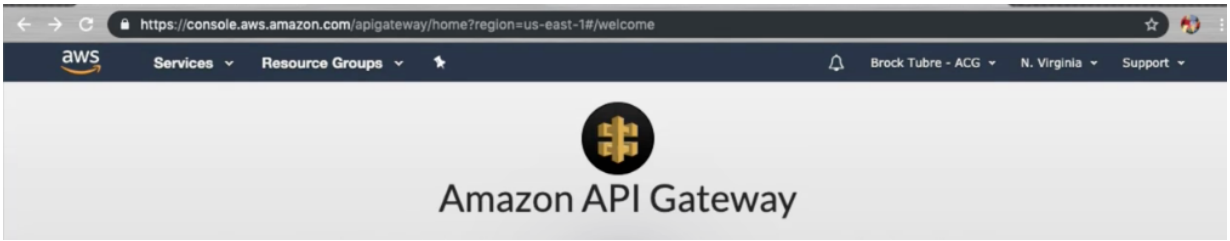


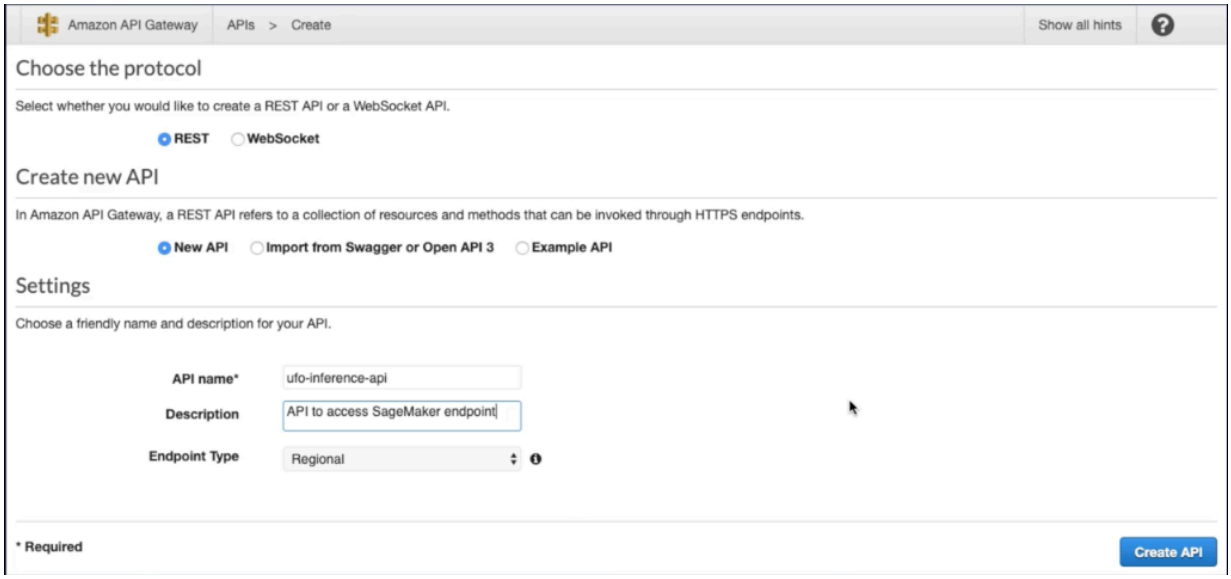
Next create an Environment variable “ENDPOINT_NAME” and add the pagemaker url to our endpoint



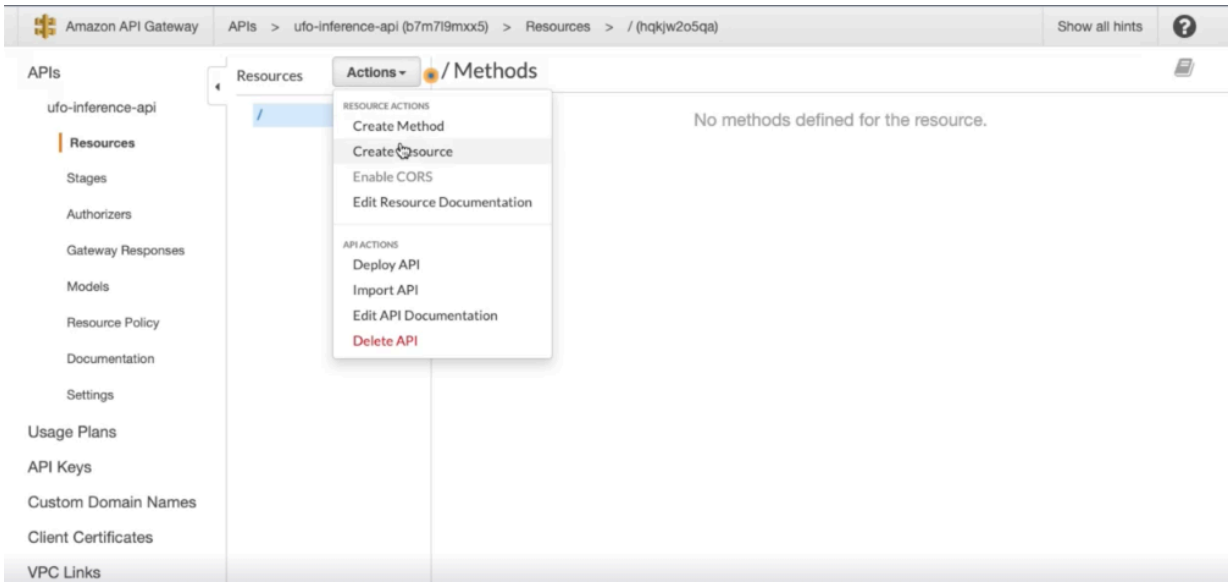


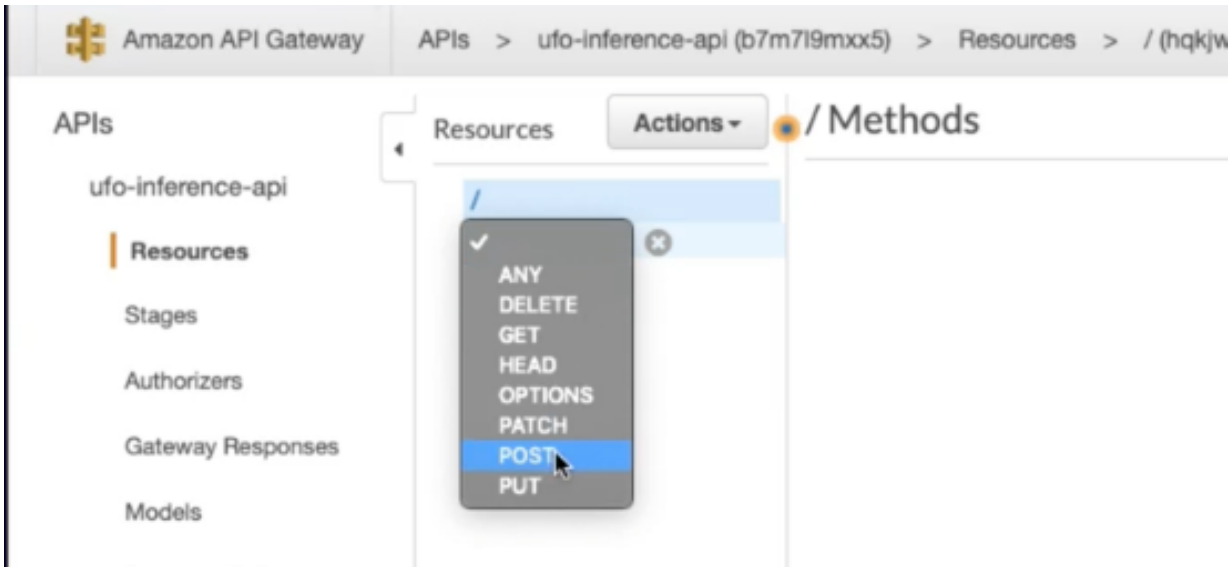
Next Create an API gateway endpoint which will call the lambda
It will have a simple post method

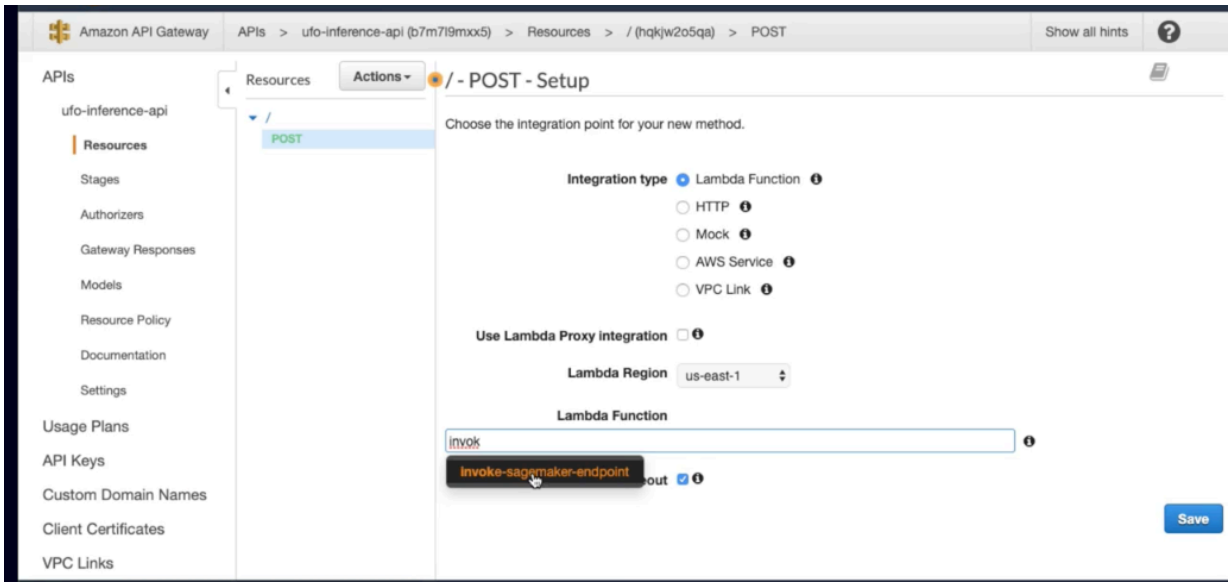




Create method => Post

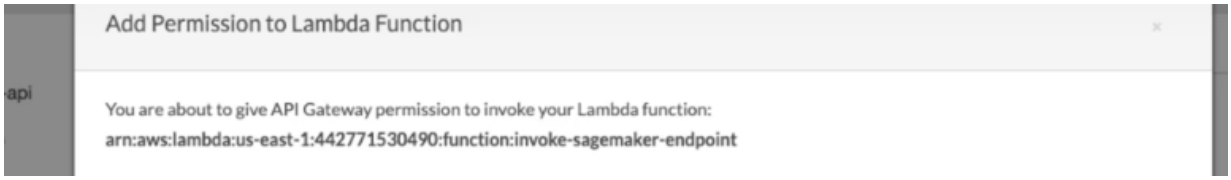

Amazon API Gateway
APIs > ufo-inference-api (b7m7l9mxx5) > Resources > / (hqkjw2o5qa)
Show all hints
APIs
ufo-inference-api
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Settings
Usage Plans
API Keys
Custom Domain Names
Client Certificates
VPC Links
Resources
Actions
/ Methods
RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation
API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API
No methods defined for the resource.

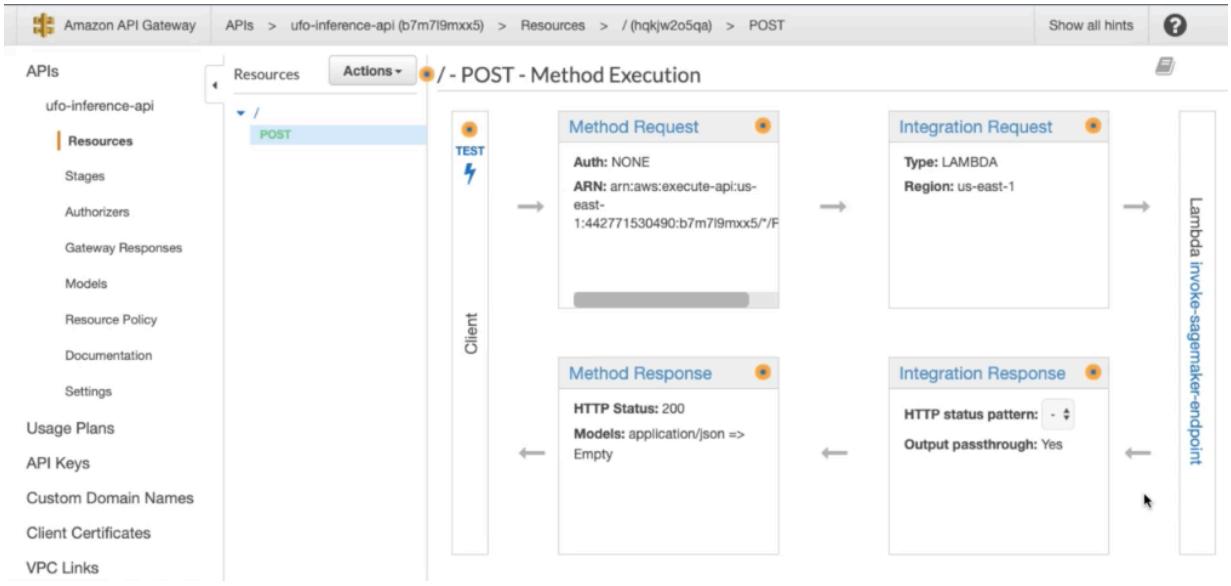

Amazon API Gateway
APIs > ufo-inference-api (b7m7l9mxx5) > Resources > / (hqkjw
APIs
ufo-inference-api
Resources
Stages
Authorizers
Gateway Responses
Models
Resources
Actions
/ Methods
/
ANY
DELETE
GET
HEAD
OPTIONS
PATCH
POST
PUT



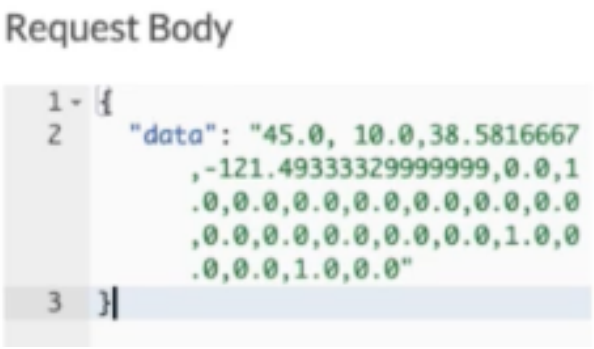
Now API gateway needs to set permissions to allow access to invoke the lambda function => select ok

\

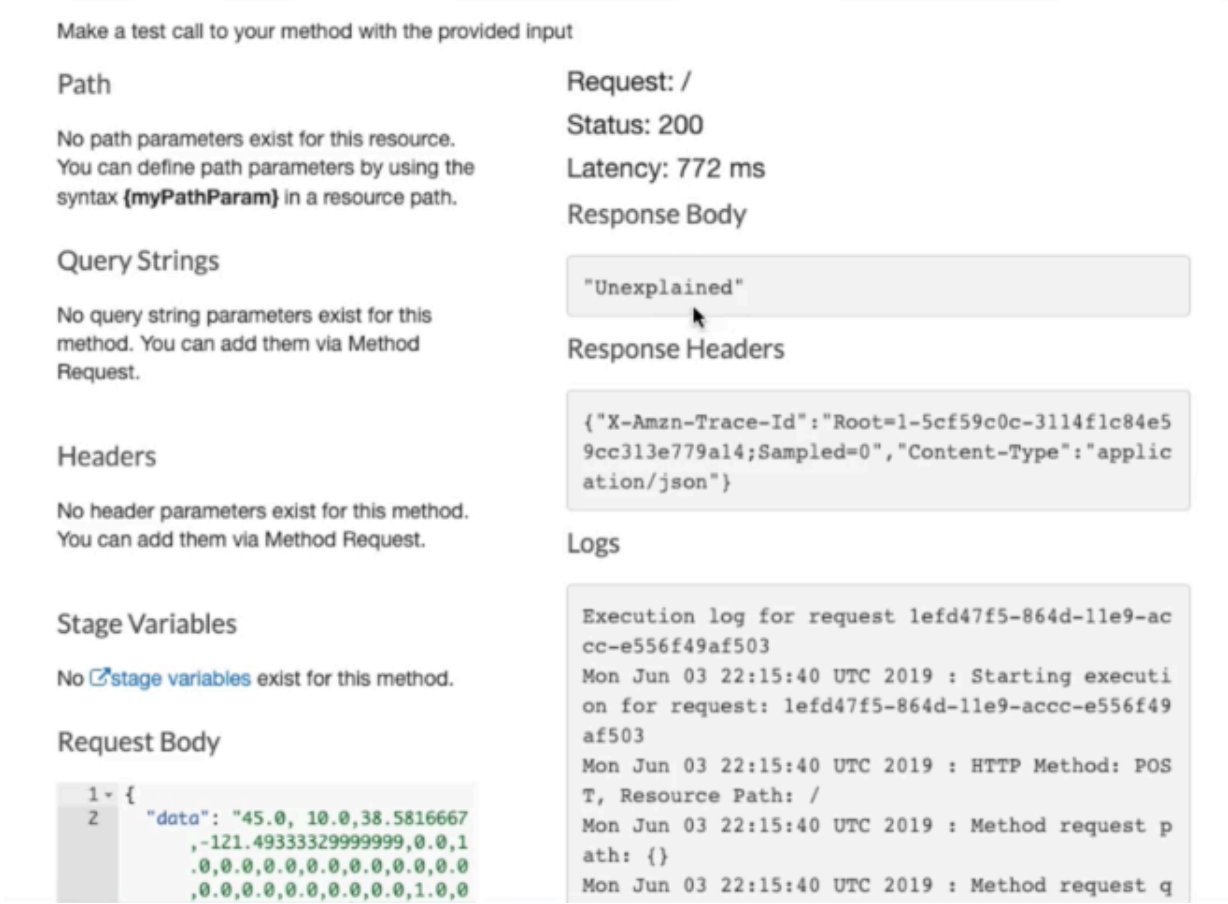




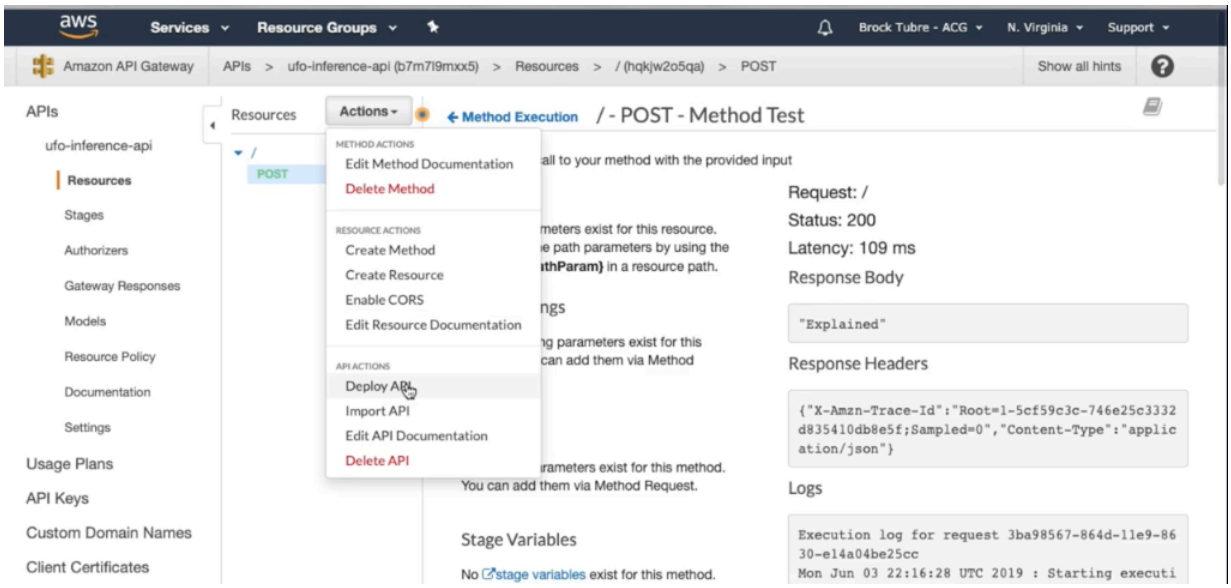
Now test with sample json request

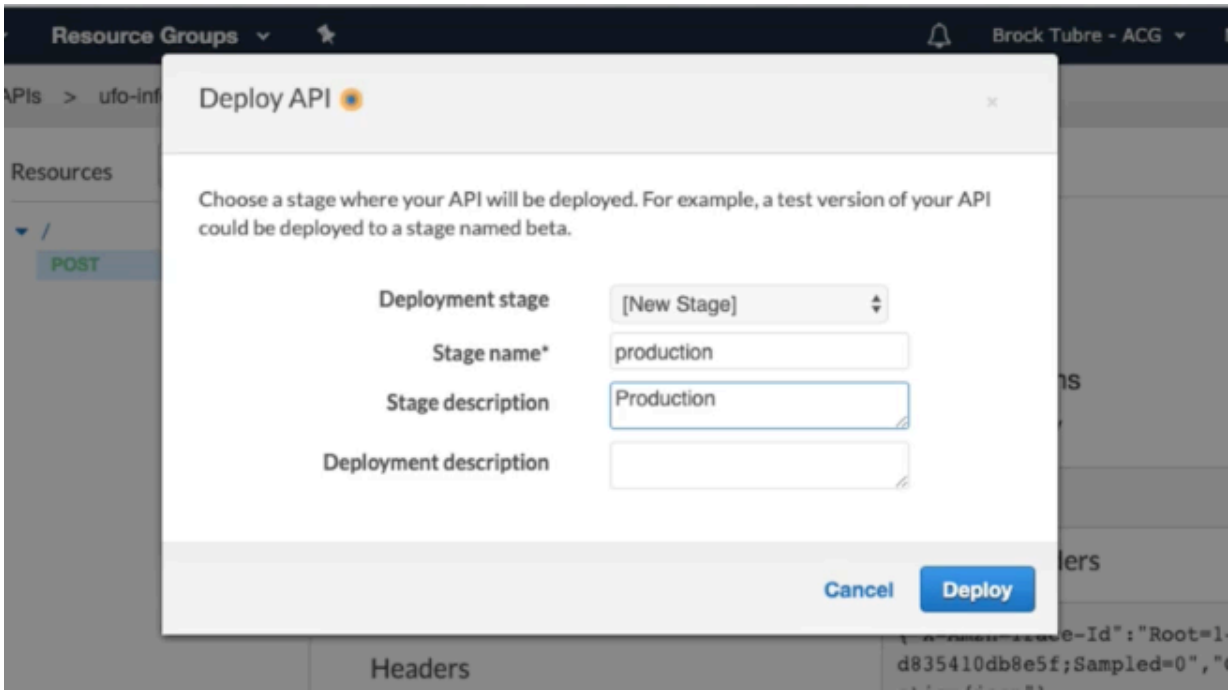


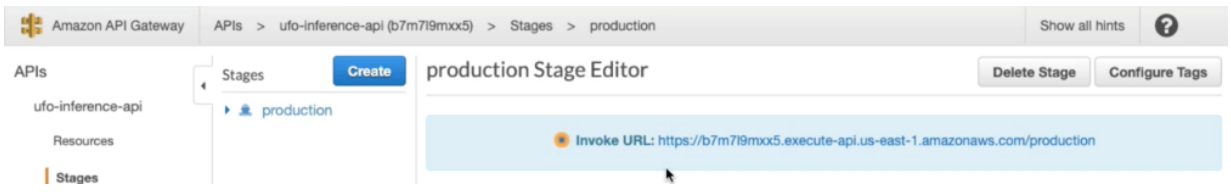
And see the response



Now we can launch this API to production -> deploy API


aws
Services
Resource Groups
Brock Tubre - ACG
N. Virginia
Support
Amazon API Gateway
APIs > ufo-inference-api (b7m7l9mxx5) > Resources > / (hqkjw2o5qa) > POST
Show all hints
APIs
ufo-inference-api
Resources
Stages
Authorizers
Gateway Responses
Models
Resource Policy
Documentation
Settings
Usage Plans
API Keys
Custom Domain Names
Client Certificates
Resources
Actions
Method Execution
/ - POST - Method Test
METHOD ACTIONS
Edit Method Documentation
Delete Method
RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation
API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API
POST
Request: /
Status: 200
Latency: 109 ms
Response Body
"Explained"
Response Headers
{"X-Amzn-Trace-Id":"Root=1-5cf59c3c-746e25c3332d835410db8e5f;Sampled=0","Content-Type":"application/json"}
Logs
Execution log for request 3ba98567-864d-11e9-8630-e14a04be25cc
Mon Jun 03 22:16:28 UTC 2019 : Starting executi
Stage Variables
No stage variables exist for this method.


Resource Groups
Brock Tubre - ACG
Deploy API
Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.
Deployment stage
[New Stage]
Stage name*
production
Stage description
Production
Deployment description
Cancel
Deploy
Resources
POST
Headers


Amazon API Gateway
APIs > ufo-inference-api (b7m7l9mxx5) > Stages > production
Show all hints
APIs
ufo-inference-api
Resources
Stages
Stages
Create
production
production Stage Editor
Delete Stage
Configure Tags
Invoke URL: https://b7m7l9mxx5.execute-api.us-east-1.amazonaws.com/production