

# Cloud Guru - 8 - Implementation and Operations Lab

**Deploy Model into Production**

Mr. K has given us the go to deploy the optimized Linear Learner model in production! Once this is complete Mr. K's team can use it to investigate any newly reported UFO sighting.

Our goal is to deploy the model into production and give Mr. K's team some way to interact with the deployed model.



A Cloud Guru logo

- Deploy our Linear Learner model using SageMaker hosting.
- Create a way to interact with the SageMaker endpoint created for the deployed model.
- How will Mr. K's team interact with the model? Will there be some type of user interface?
- What does the input and output of a request look like? Will it be in batches or an immediate response?

- Make a request to the SageMaker hosted model
- View the output response from the trained model

Explained  
Unexplained  
Probable

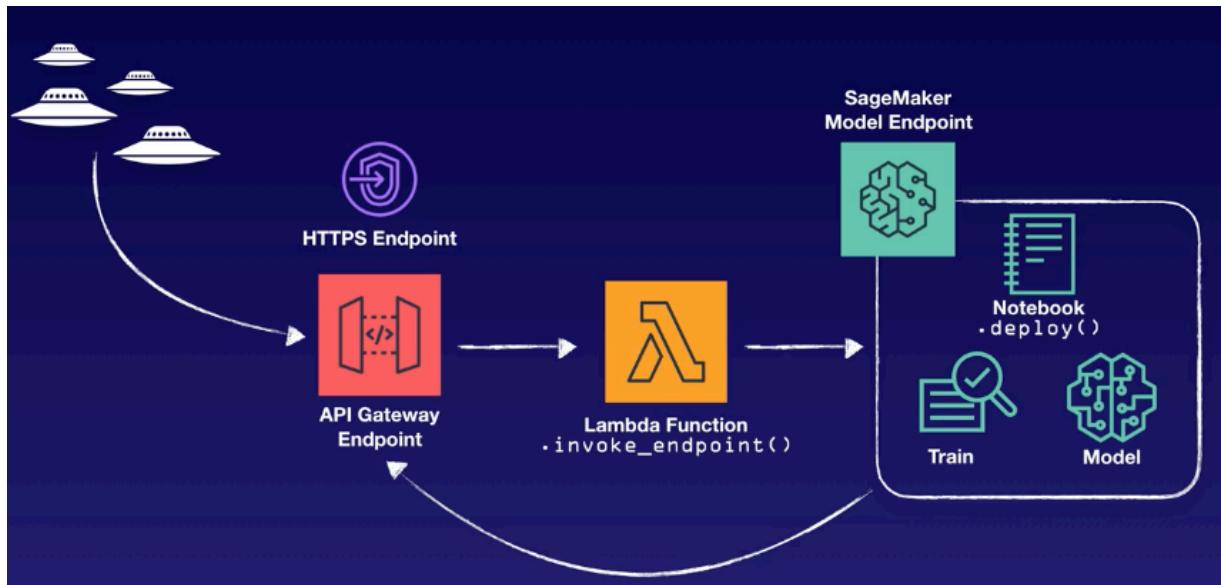
```
{  
  "data":  
    "45.0, 10.0, 38.5816667,  
    -121.4933332999999,  
    0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,  
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  
    1.0, 0.0, 0.0, 1.0, 0.0"  
}
```

Simply call the `.deploy()` method after training our model to deploy it to SageMaker hosting.

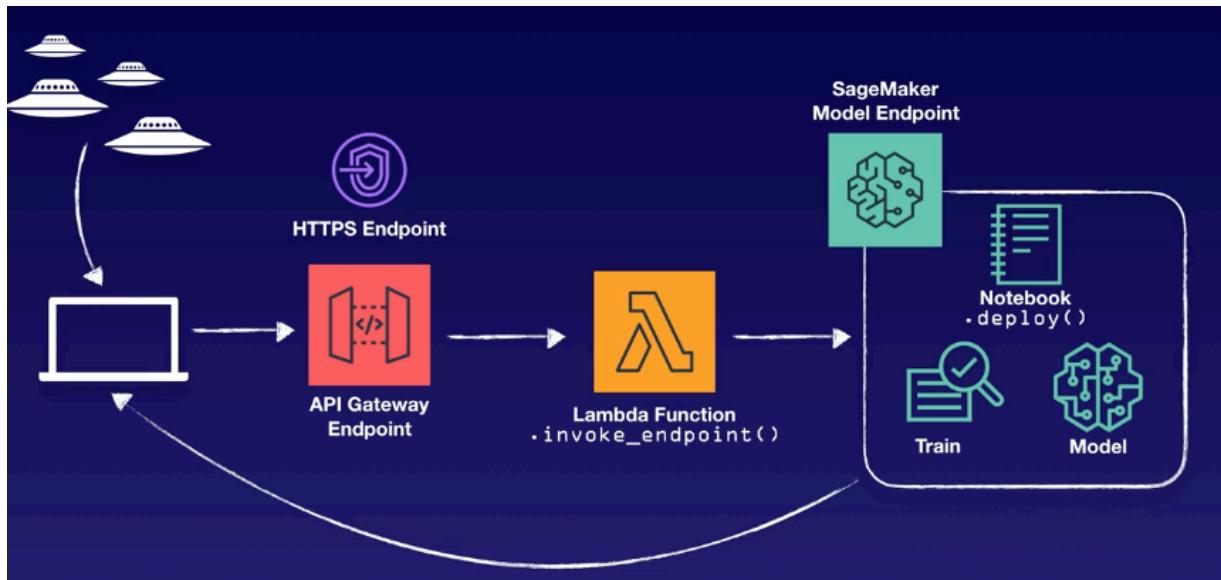
=> just one line of code in Jupiter

After deployment:

Create an API Gateway endpoint that invokes a Lambda function. This Lambda function calls the `.invoke_endpoint()` method with any new UFO sighting. Return the results from the models prediction.



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

```
In [ ]: import pandas as pd
import numpy as np
from datetime import datetime
import io
import sagemaker.amazon.common as smac

import boto3
from sagemaker import get_execution_role
import sagemaker

import matplotlib.pyplot as plt
```

## 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	1977-04-04T04:23.340Z	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	Watcica	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 type.
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` because it is always 'Y' or Yes.
2. Let's drop the `firstName` and `lastName` because they are not important in determining the `researchOutcome`.
3. Let's drop the `reportedTimestamp` because 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 algorithm 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_mostlyCloudy	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 Learner 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(train_recordIO_protobuf_location = 's3://{}/implementation_operations_lab/linearlearner_train/{}'.format(bucket, train_recordIO_protobuf_location)))
print('The Pipe mode recordIO protobuf training data: {} format(recordIO protobuf location)'.format(train_recordIO_protobuf_location))

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_validation_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, validate_recordIO_protobuf_location)))
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_validation/ufo_sightings_validation_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 LinearLearner 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 hyperparameters 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 LinearLearner 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,
                            lr=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_name))
```

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": 1, "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_f1_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>=0.93908629441624369
[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>=0.97582009493531441
[06/03/2019 20:08:54 INFO 139948801308480] #quality_metric: host=algo-1, validation macro_recall <score>=0.9268350000000000

```

```

[06/03/2019 20:08:54 INFO 139948801308480] test data is not provided.
[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}, "setup.time": {"count": 1, "max": 14.787912368774414, "sum": 14.787912368774414, "min": 14.787912368774414}, "update.time": {"count": 11, "max": 829.30118850708, "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]: https://s3.amazonaws.com/ml-labs-acg/implementation_operations_lab/linearlearner_output/ufo-linear-learner-job-20190603200343/output/model.tar.gz

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

The screenshot shows the AWS SageMaker console interface. On the left, a sidebar lists various resources under categories like Labeling workforces, Notebook, Training, Inference, and Endpoints. The 'Endpoints' item is selected. The main area displays a table of endpoints with columns for Name, ARN, Creation time, Status, and Last updated. One endpoint is listed:

Name	ARN	Creation time	Status	Last updated
ufo-linear-learner-job-20190603200343	arn:aws:sagemaker:us-east-1:442771530490:endpoint/ufo-linear-learner-job-20190603200343	Jun 03, 2019 20:36 UTC	InService	Jun 03, 2019 20:45 UTC

Clicking on the endpoint name leads to its detailed configuration page. This page includes sections for Endpoint settings, View invocation metrics, View instance metrics, View logs, and a metrics chart.

**Endpoint settings:**

- Name:** ufo-linear-learner-job-20190603200343
- ARN:** arn:aws:sagemaker:us-east-1:442771530490:endpoint/ufo-linear-learner-job-20190603200343
- Status:** InService

**Metrics:**

- DiskUtilization:** Shows a sharp peak at approximately 0.84 around 20:37 on June 3rd.
- CPUUtilization:** Shows a sharp peak at approximately 0.046 around 20:37 on June 3rd.
- MemoryUtilization:** Shows a sharp peak at approximately 0.555 around 20:37 on June 3rd.

Endpoint runtime settings								
			Update weights		Update instance count		Configure auto scaling	
Variant name			Instance type	Elastic Inference	Current instance count	Desired instance count	Instance min - max	
AllTraffic	1	1	ml.m4.xlarge	-	1	1	-	-

ARN	Creation time					
arn:aws:sagemaker:us-east-1:442771530490:endpoint-config/ufo-linear-learner-job-20190603200343	Jun 03, 2019 20:36 UTC					
<b>Production variants</b>						
Model name	Training job	Variant name	Instance type	Elastic Inference	Initial instance count	Initial weight
<a href="#">linear-learner-2019-06-03-20-36-46-643</a>	<a href="#">ufo-linear-learner-job-20190603200343</a>	AllTraffic	ml.m4.xlarge	-	1	1

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=plt.cm.Greens):
    """
    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'
        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)

```

```

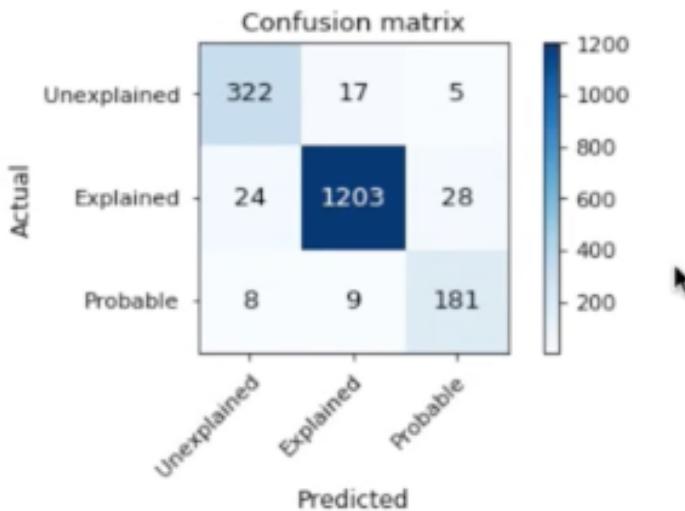
In [ ]: %matplotlib inline
sns.set_context("paper", font_scale=1.4)

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]))

```

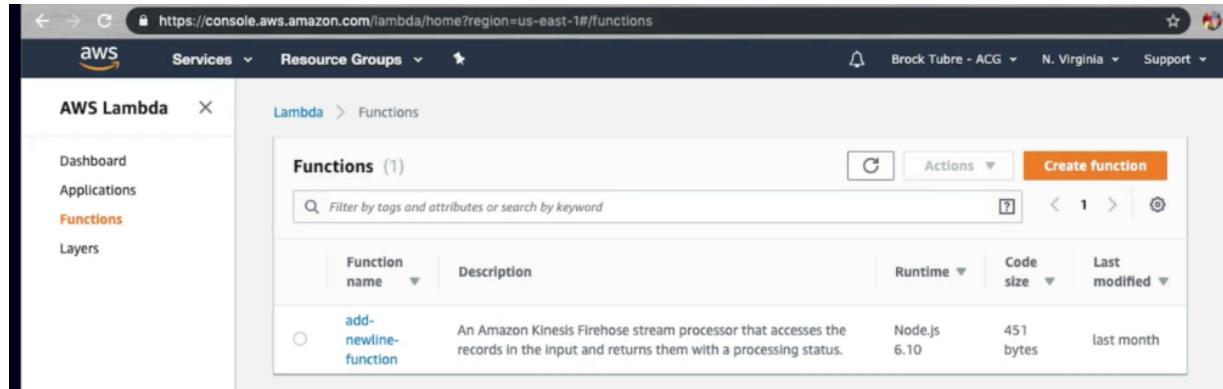
```

Accuracy is: 0.949360445186422
Precision is: 0.9114145004647342
Recall is: 0.9362512209403713
F1 score is: 0.9232919712224389

```

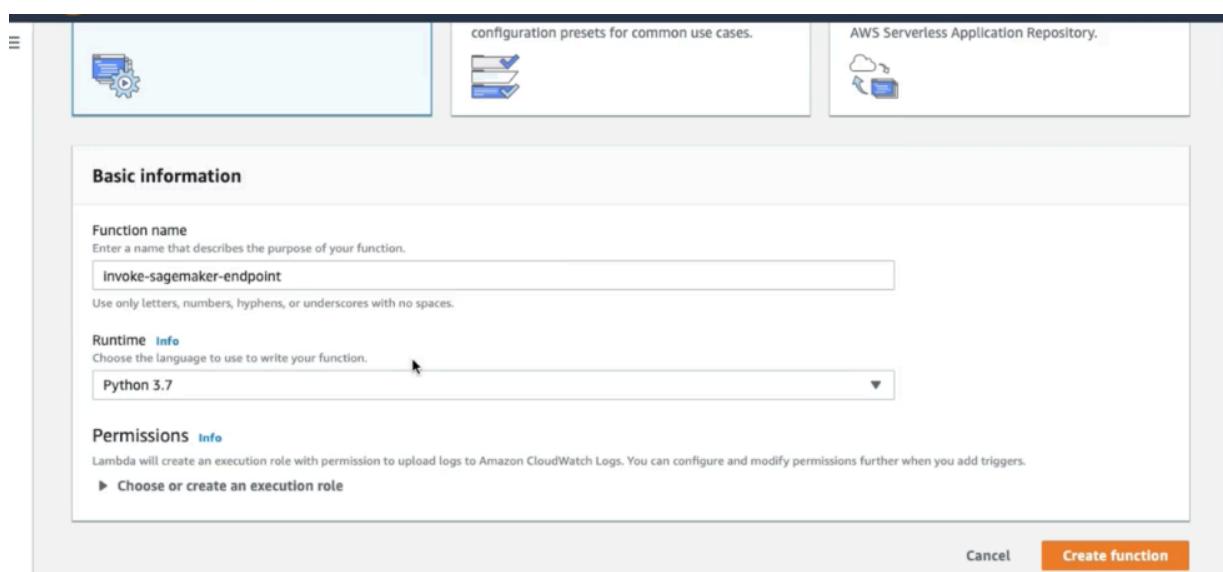
Our model is actually a little more accurate on our test data than on our training data.

Next create a lambda



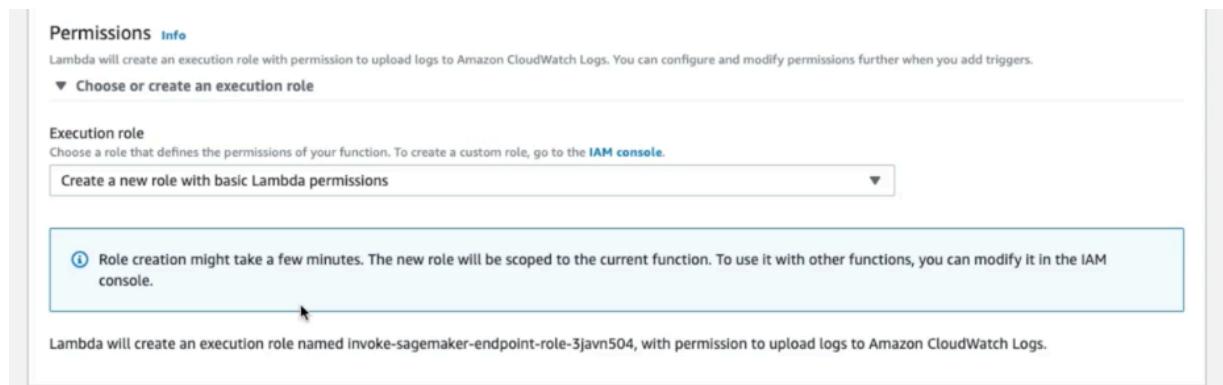
The screenshot shows the AWS Lambda Functions page. The left sidebar has 'Functions' selected. The main area displays a table with one row:

Function name	Description	Runtime	Code size	Last modified
add-newline-function	An Amazon Kinesis Firehose stream processor that accesses the records in the input and returns them with a processing status.	Node.js 6.10	451 bytes	last month

The screenshot shows the 'Basic information' step of the 'Create function' wizard. It includes fields for Function name (set to 'Invoke-sagemaker-endpoint'), Runtime (set to 'Python 3.7'), and Permissions (with a note about creating an execution role).

Create an IAM role that will have access to SageMaker



The screenshot shows the 'Permissions' step of the IAM wizard. It includes sections for choosing an execution role (with a note about creating a new role), selecting an execution role (with a dropdown for 'Create a new role with basic Lambda permissions'), and a note about role creation taking a few minutes. At the bottom, it states that a new execution role will be created with log upload permissions.

Now go to IAM and give access to SageMaker

The screenshot shows the AWS IAM service page. At the top, there's a navigation bar with 'Services' (dropdown), 'Resource Groups' (dropdown), and account information ('Brock Tuber - ACG', 'N. Virginia', 'Support'). Below the navigation is a search bar containing the text 'IAM'. To the right of the search bar are buttons for 'Group' and 'A-Z'. On the left, a sidebar lists services: History, Lambda, Amazon SageMaker, Console Home, API Gateway, Cloud9, EC2, Lightsail, ECR, ECS, Athena, AWS RoboMaker, EMR, CloudSearch, Blockchain, Elasticsearch Service, Alexa for Business, Amazon Chime, and WorkMail. The 'Amazon SageMaker' service is highlighted.

Fine the role we just created:

The screenshot shows the 'Create role' page in the IAM console. On the left, there's a sidebar with 'Policies', 'Identity providers', 'Account settings', 'Credential report', 'Encryption keys', and a 'Search' bar with the text 'sage'. The main area has tabs for 'Create role' (selected) and 'Delete role'. It includes a search bar and a table with two results. The first result is 'AmazonSageMaker-ExecutionRole' with a description: 'SageMaker execution role created from the SageMaker AWS Lambda template'. The second result is 'invoke-sagemaker-endpoint-role-3javn504' with a description: 'Invoke-sagemaker-endpoint-role-3javn504'. Both rows show 'AWS service: sagemaker' under 'Trusted entities'.

Attach a policy with full access to SageMaker

Add permissions to invoke-sagemaker-endpoint-role-3javn504

Attach Permissions

The screenshot shows the 'Attach Permissions' dialog. It has a 'Create policy' button and a search bar with 'sagem'. Below is a table with four policy entries. The second row, 'AmazonSageMakerFullAccess', is selected with a checked checkbox and highlighted in blue. The other three policies are customer managed and AWS managed types with no checkboxes checked. At the bottom right are 'Cancel' and 'Attach policy' buttons.

	Policy name	Type	Used as	Description
<input type="checkbox"/>	AmazonSageMaker-ExecutionRole	Customer managed	Permissions policy (1)	
<input checked="" type="checkbox"/>	AmazonSageMakerFullAccess	AWS managed	Permissions policy (1)	Provides full access to Amazon SageMaker via the AWS Management Console.
<input type="checkbox"/>	AmazonSageMakerReadOnly	AWS managed	None	Provides read only access to Amazon SageMaker via the AWS Management Console.
<input type="checkbox"/>	AWSGlueConsoleSageMaker	AWS managed	None	Provides full access to AWS Glue via the AWS Management Console.

Now we have permission to interact with that pagemaker endpoint

Summary

Role ARN: arn:aws:iam::442771530490:role/service-role/invoke-sagemaker-endpoint-role-3javn504

Role description: Edit

Instance Profile ARNs: [Edit](#)

Path: /service-role/

Creation time: 2019-06-03 17:33 EDT

Maximum CLI/API session duration: 1 hour [Edit](#)

**Permissions** [Attach policies](#) [Add inline policy](#)

▼ Permissions policies (2 policies applied)

Policy name	Policy type
AmazonSageMakerFullAccess	AWS Policy
AmazonSageMakerInvokeFullAccess	AWS Policy

Back to Lambda, insert the code to use

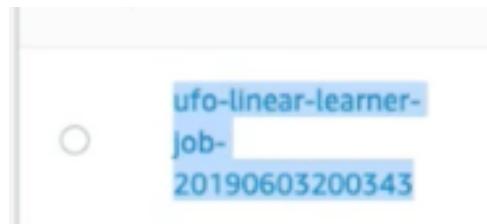
```

invoke-sagemaker-endpoint Throttle Qualifiers Actions Select a test event Test Save
Environment invoke-sagemaker-i lambda_function.py

lambda_function.py
8 ENDPOINT_NAME = os.environ['ENDPOINT_NAME']
9 runtime= boto3.client('runtime.sagemaker')
10
11 def lambda_handler(event, context):
12     print("Received event: " + json.dumps(event, indent=2))
13
14     data = json.loads(json.dumps(event))
15     payload = data['data']
16     print(payload)
17
18     response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
19                                         ContentType='text/csv',
20                                         Body=payload)
21     print(response)
22     result = json.loads(response['Body'].read().decode())
23     print(result)
24     pred = int(result['predictions'][0]['predicted_label'])
25
26     if(pred == 0):
27         return 'Unexplained'
28     if(pred == 1):
29         return 'Explained'
30     if(pred == 2):
31         return 'Probable'

```

Next create an Environment variable "ENDPOINT\_NAME" and add the pagemaker url to our endpoint



**Environment variables**

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

ENDPOINT_NAME	ufo-linear-learner-job-20190603200343	Remove
Key	Value	Remove

▶ Encryption configuration

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

The screenshot shows the AWS API Gateway 'Create new API' interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, and account information ('Brock Tube - ACG', 'N. Virginia', 'Support'). Below the navigation is the 'Amazon API Gateway' logo.

**Choose the protocol**

Select whether you would like to create a REST API or a WebSocket API.

REST    WebSocket

**Create new API**

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API    Import from Swagger or Open API 3    Example API

**Settings**

Choose a friendly name and description for your API.

API name\*   
Description   
Endpoint Type

\* Required

Create method => Post

Amazon API Gateway   APIs > ufo-inference-api (b7m7l9mxx5) > Resources > / (hqkjw2o5qa)   Show all hints ?

APIs   Resources   Actions / Methods

No methods defined for the resource.

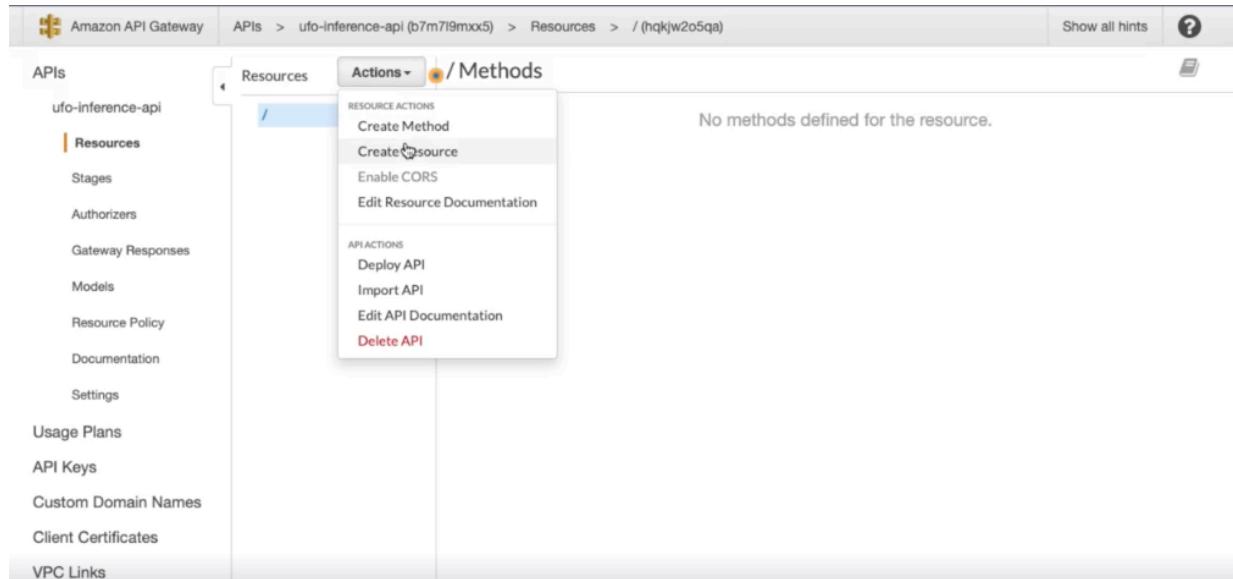
RESOURCES ACTIONS

- Create Method
- Create  source**
- Enable CORS
- Edit Resource Documentation

API ACTIONS

- Deploy API
- Import API
- Edit API Documentation
- Delete API

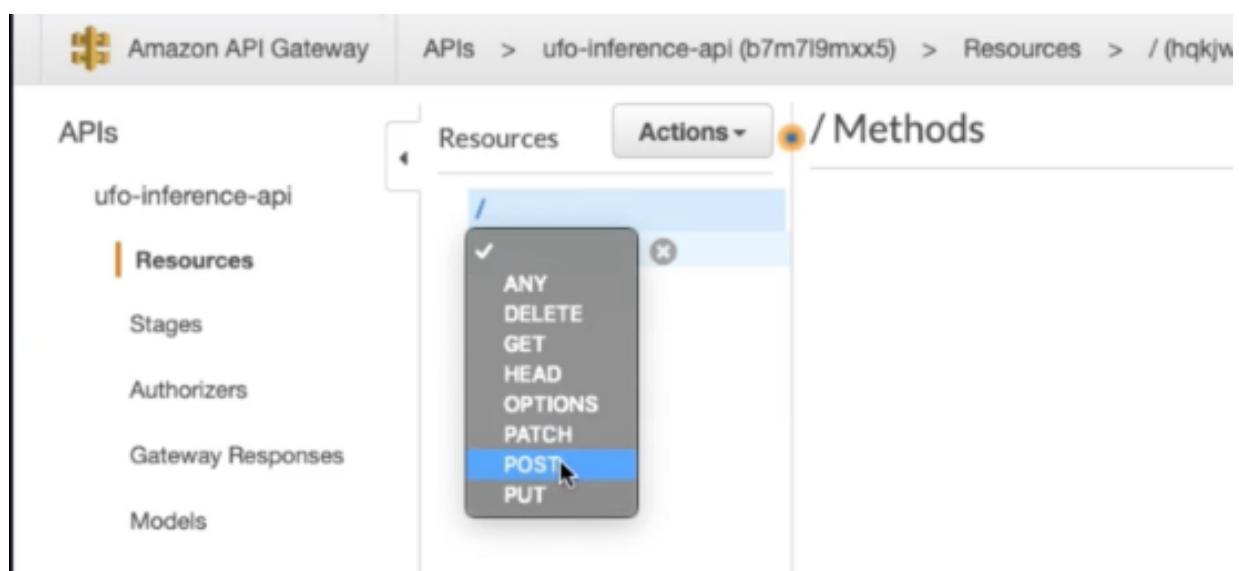
API Keys   Custom Domain Names   Client Certificates   VPC Links

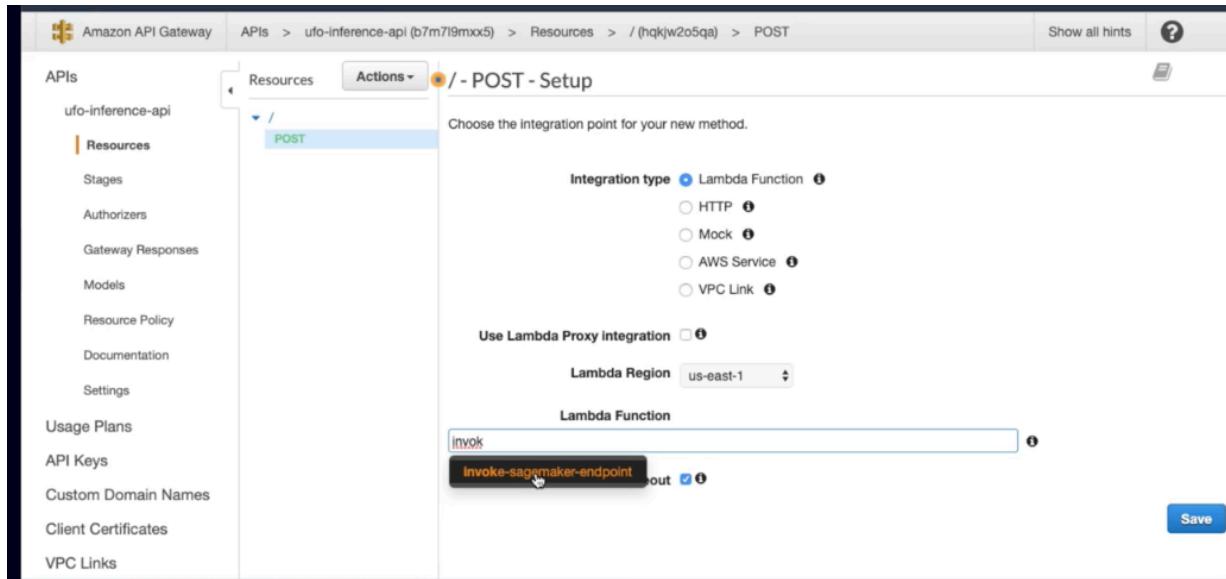


Amazon API Gateway   APIs > ufo-inference-api (b7m7l9mxx5) > Resources > / (hqkjw2o5qa)

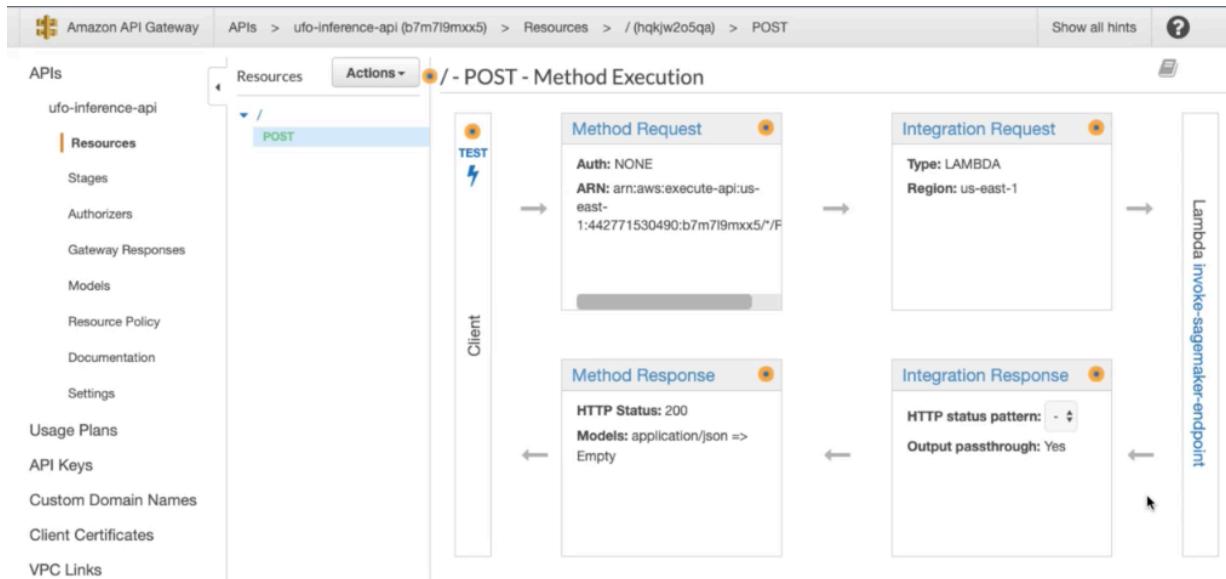
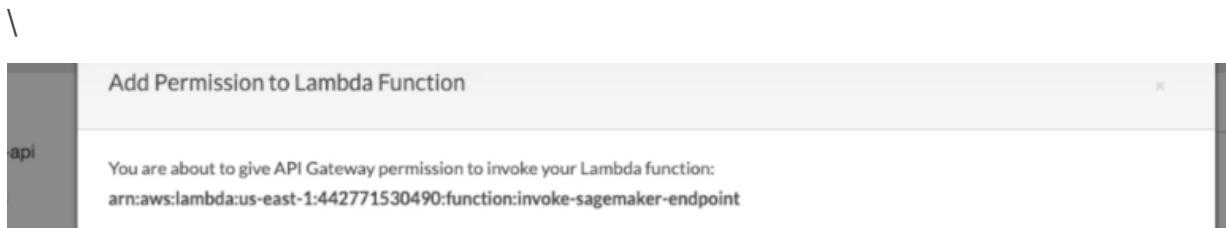
APIs   Resources   Actions / Methods

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

## Request Body

```
1 - {
2   "data": "45.0, 10.0,38.5816667
,-121.4933332999999,0.0,1
.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,1.0,0
.0,0.0,1.0,0.0"
3 }
```

And see the response

Make a test call to your method with the provided input

### Path

No path parameters exist for this resource.  
You can define path parameters by using the syntax **{myPathParam}** in a resource path.

Request: /

Status: 200

Latency: 772 ms

### Response Body

```
"Unexplained"
```

### Query Strings

No query string parameters exist for this method. You can add them via Method Request.

### Response Headers

```
{"X-Amzn-Trace-Id":"Root=1-5cf59c0c-3114f1c84e5
9cc313e779a14;Sampled=0","Content-Type":"appli
cation/json"}
```

### Headers

No header parameters exist for this method. You can add them via Method Request.

### Logs

```
Execution log for request lefd47f5-864d-11e9-ac
cc-e556f49af503
Mon Jun 03 22:15:40 UTC 2019 : Starting executi
on for request: lefd47f5-864d-11e9-accc-e556f49
af503
Mon Jun 03 22:15:40 UTC 2019 : HTTP Method: POS
T, Resource Path: /
Mon Jun 03 22:15:40 UTC 2019 : Method request p
ath: {}
Mon Jun 03 22:15:40 UTC 2019 : Method request q
```

### Stage Variables

No **stage variables** exist for this method.

### Request Body

```
1 - {
2   "data": "45.0, 10.0,38.5816667
,-121.4933332999999,0.0,1
.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,1.0,0
.0,0.0,1.0,0.0"
3 }
```

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

Screenshot of the AWS Amazon API Gateway Method Execution page for a POST method.

The left sidebar shows the API structure:

- APIs
- ufo-inference-api
- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Settings
- Usage Plans
- API Keys
- Custom Domain Names
- Client Certificates

The main panel shows the Method Execution details for the POST method:

- Actions**:
  - Edit Method Documentation
  - Delete Method
- METHOD ACTIONS**:
  - Create Method
  - Create Resource
  - Enable CORS
  - Edit Resource Documentation
- RESOURCE ACTIONS**:
  - Deploy API
  - Import API
  - Edit API Documentation
  - Delete API
- API ACTIONS**:
  - [disabled]

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 execution

Screenshot of the AWS Amazon API Gateway Deploy API dialog.

The dialog title is "Deploy API".

Instructions: 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: (empty)

Buttons: Cancel, Deploy

Screenshot of the AWS Amazon API Gateway Stages page.

The left sidebar shows the API structure:

- APIs
- ufo-inference-api
- Resources
- Stages

The main panel shows the Stage Editor for the production stage:

- Create button
- production Stage Editor
- Delete Stage button
- Configure Tags button

Invoke URL: <https://b7m7l9mxx5.execute-api.us-east-1.amazonaws.com/production>