Deploying ML Models on AWS platform

Agenda

- Train a model locally. (Local environment should be ready).
- Upload the model to AWS S3 bucket.
- Create a lambda function.
- Integrate with API gateway.
- Optional (discussion on layers)
- Optional (discussion on how to productionize DNN models on Lambda)

INTRODUCTIONS

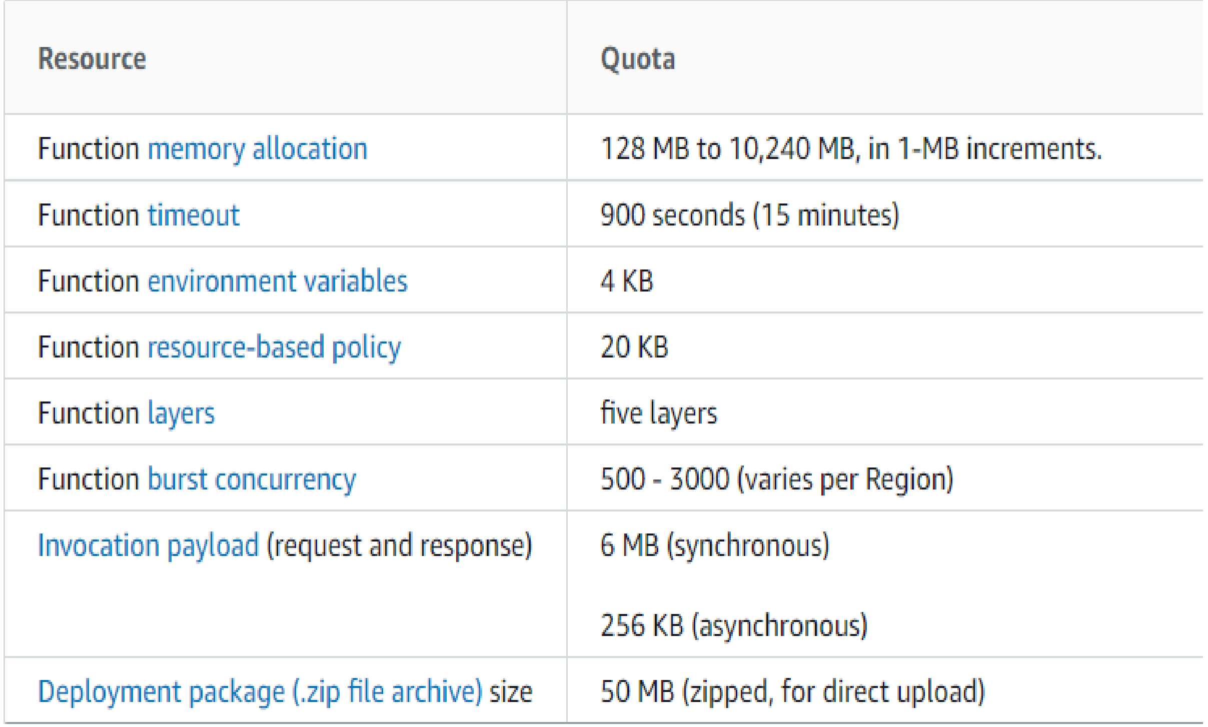
A Request

- This will be a hands-on session
- Please do not ask code for copy paste. This exercise to make you implement a ml model, end to end, copy paste will not serve the purpose.
- Please remain actively engaged during workshop and follow facilitator’s guidance·
- Please keep your video on for the entire session duration
- Feedback link available below must be filled. That will be also the record of your participation in this session.
- Do ask for breaks.

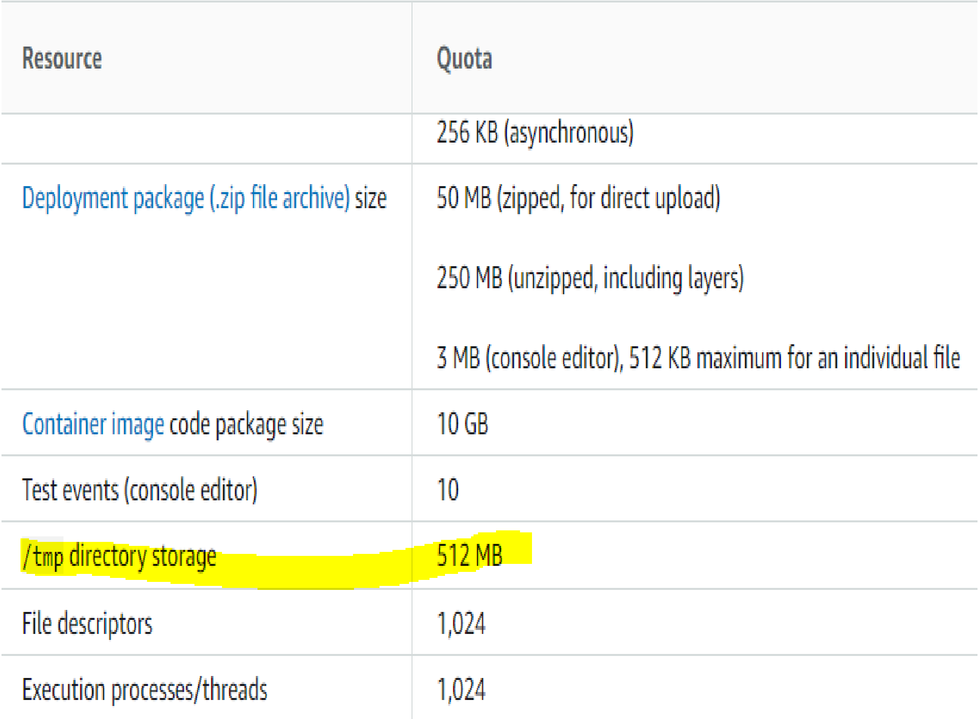
Features of AWS Lambda

- For hosting small models
- Serverless
- Suitable for Low-cost scenario, experimentation. When you need to have a end to end demo.
- No GPU
- Deep Neural Networks of reasonable size can be productionized.
- Suitable for domains like Manufacturing.
- No heavy infrastructure, less maintenance.
- ML Pipeline can be established (retraining) – Can address some of the ML lifecycle challenges.

Lambda Limits



Resource	Quota
Function memory allocation	128 MB to 10,240 MB, in 1-MB increments.
Function timeout	900 seconds (15 minutes)
Function environment variables	4 KB
Function resource-based policy	20 KB
Function layers	five layers
Function burst concurrency	500 - 3000 (varies per Region)
Invocation payload (request and response)	6 MB (synchronous) 256 KB (asynchronous)
Deployment package (.zip file archive) size	50 MB (zipped, for direct upload)



Resource	Quota
	256 KB (asynchronous)
Deployment package (.zip file archive) size	50 MB (zipped, for direct upload) 250 MB (unzipped, including layers) 3 MB (console editor), 512 KB maximum for an individual file
Container image code package size	10 GB
Test events (console editor)	10
/tmp directory storage	512 MB
File descriptors	1,024
Execution processes/threads	1,024

Part – A - Train the model (if we do not have)

- Use Wine dataset for training a model.
- Save the model to disk.
- Load the model from the disk.
- Do prediction on sample data.

Train the model

```
from sklearn.datasets import load_wine
import pandas as pd
import numpy as np
import pickle

data = load_wine() # import dataset
df = pd.DataFrame(data['data'], columns=data['feature_names']) # build dataframe
df['target'] = data['target'] # add dependent variable
df = df.sample(frac=1) # randomize the data
df.head(3)

train_df = df[:150]
test_df = df[150:]

feature_cols = ['alcohol', 'malic_acid']
X = train_df[feature_cols] # Features
y = train_df.target # Target variable

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

Train the model

```
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
```

Testing (Predicting) for one observation:

```
data = {
    "alcohol": 11.1, "malic_acid": 5
}
df = pd.DataFrame([data])
result = logreg.predict(df)
result[0]
```

Creating Model file

```
import pickle
pickle.dump( logreg, open( "wine-model-masterclass-2-input-params.p", "wb" ))

# change the name so that it is unique to you <model-wine-ritesh.p>
```

Testing the model from disk at Training time:

- This is an important step.
- This is setting up the code which can be used at the time of setting this up.
- if you have done this step well, rest of things will be easy.

- Load_fn
- Input_fn
- Preprocessing_fn
- Predict_fn
- Output_fn

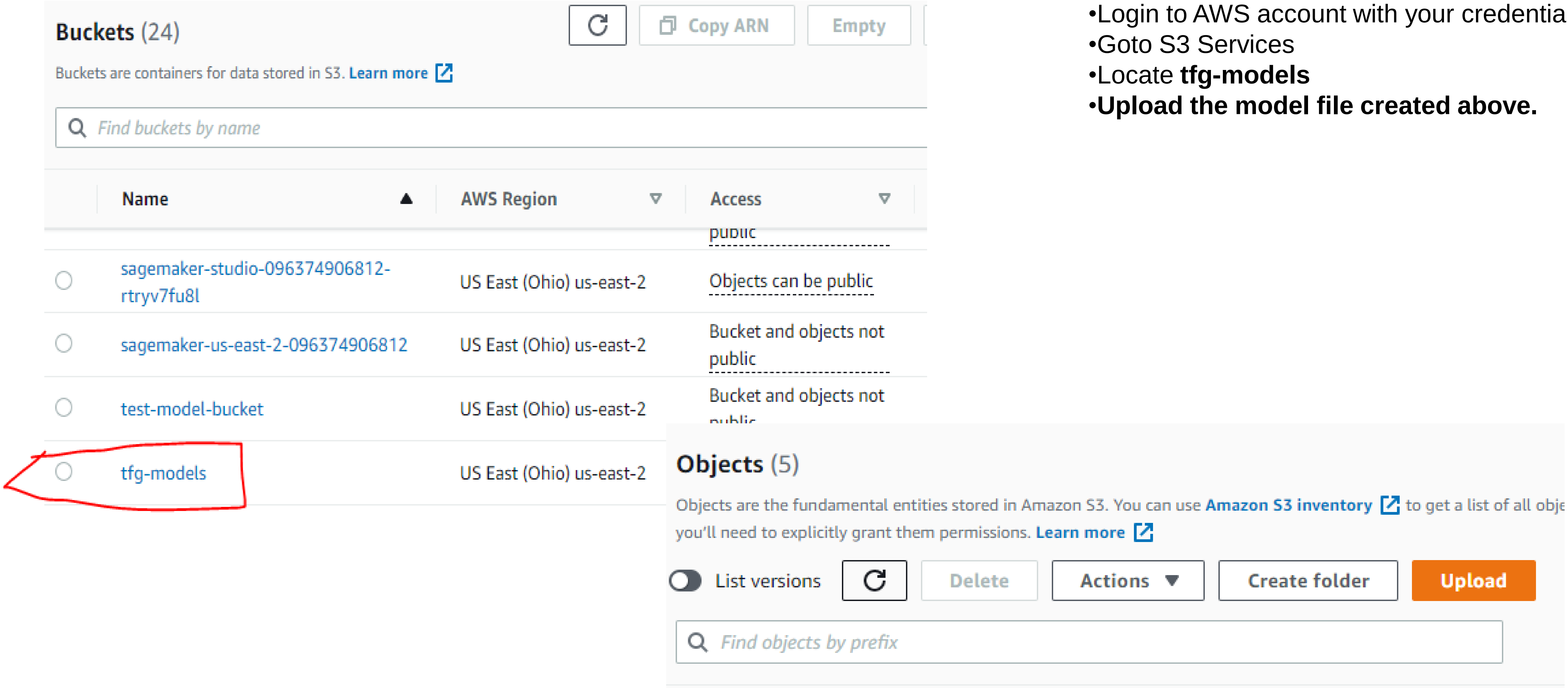
```
temp_file_path = "wine-model-masterclass-2-input-params.p"
with open(temp_file_path, 'rb') as f:
  model = pickle.load(f)

input_ = {"alcohol": 105, "malic_acid": 13}
df_input = pd.DataFrame([input_])
df_input
model.predict(df_input)
```

PART B - Build the Lambda function

- Putting the model on S3 bucket
- Add Layers (Prebuilt)
- Add Environment Variables
- Building Lambda function (Step by Step)

Upload model to AWS

- Login to AWS account with your credentials.
- Goto S3 Services
- Locate **tfg-models**
- **Upload the model file created above.**



Buckets are containers for data stored in S3. Learn more

	Name	AWS Region	Access
			public
○	sagemaker-studio-096374906812-rtryv7fu8l	US East (Ohio) us-east-2	Objects can be public
○	sagemaker-us-east-2-096374906812	US East (Ohio) us-east-2	Bucket and objects not public
○	test-model-bucket	US East (Ohio) us-east-2	Bucket and objects not public
○	tfg-models	US East (Ohio) us-east-2	

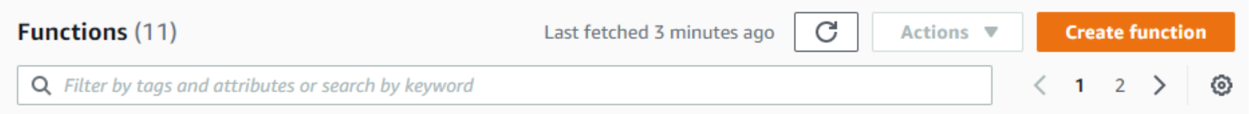
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all obje you'll need to explicitly grant them permissions. Learn more

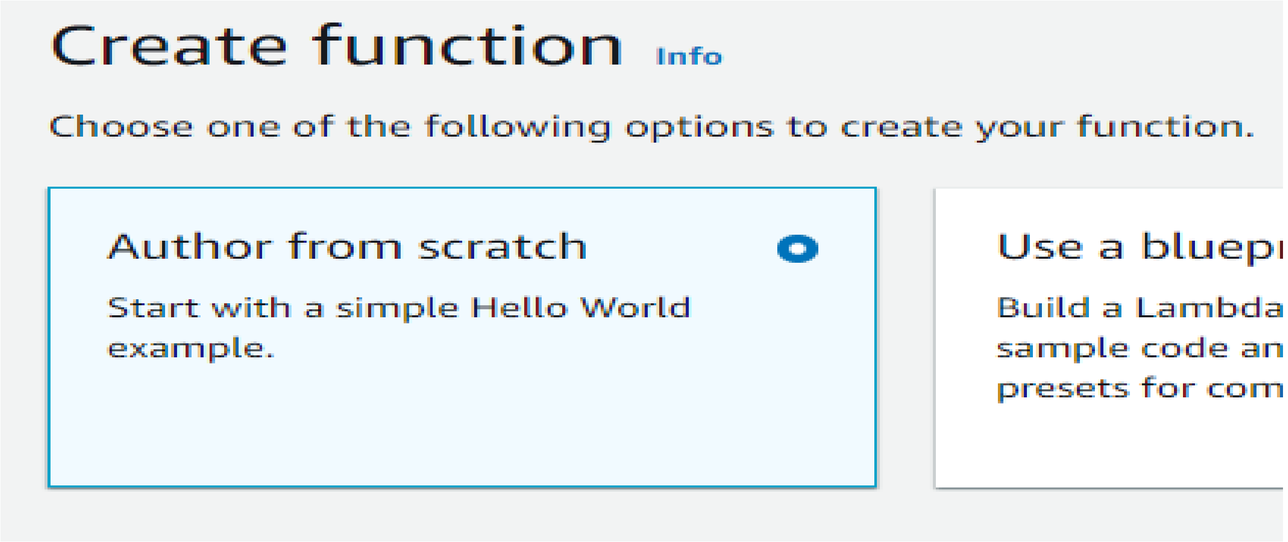
Building lambda function

Go to Lambda services

https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions

Create Function


Functions (11)
Last fetched 3 minutes ago
Actions ▼
Create function
Filter by tags and attributes or search by keyword
1
2



Choose one of the following options to create your function.

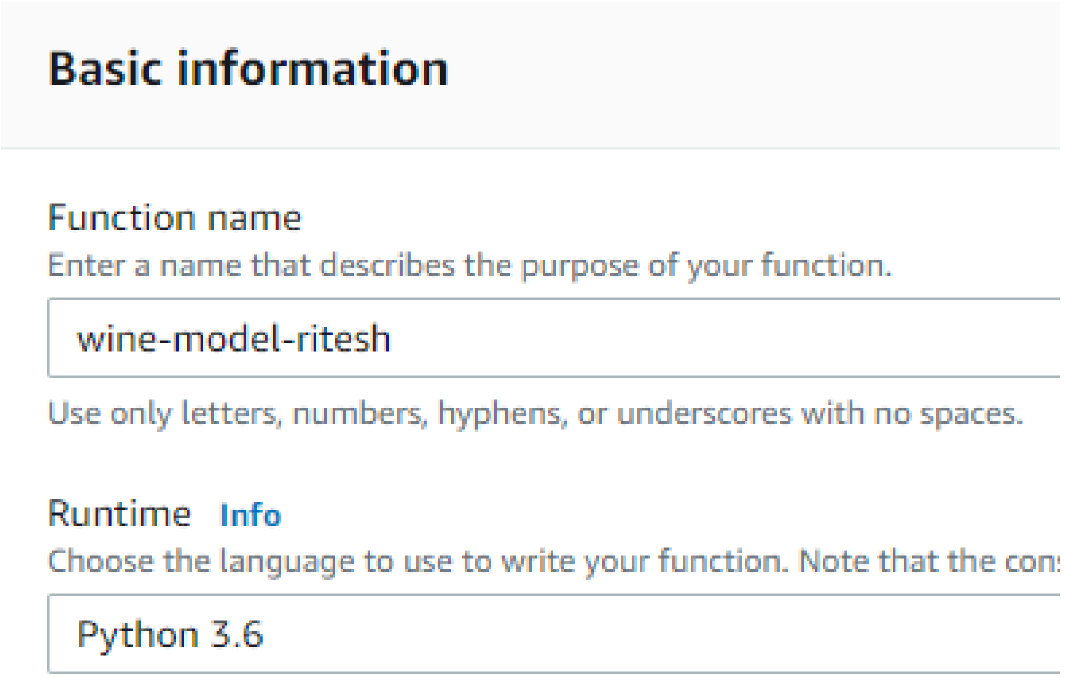
Author from scratch ●

Start with a simple Hello World example.

Use a bluepr

Build a Lambda
sample code an
presets for com

Building lambda function



Basic information

Function name

Enter a name that describes the purpose of your function.

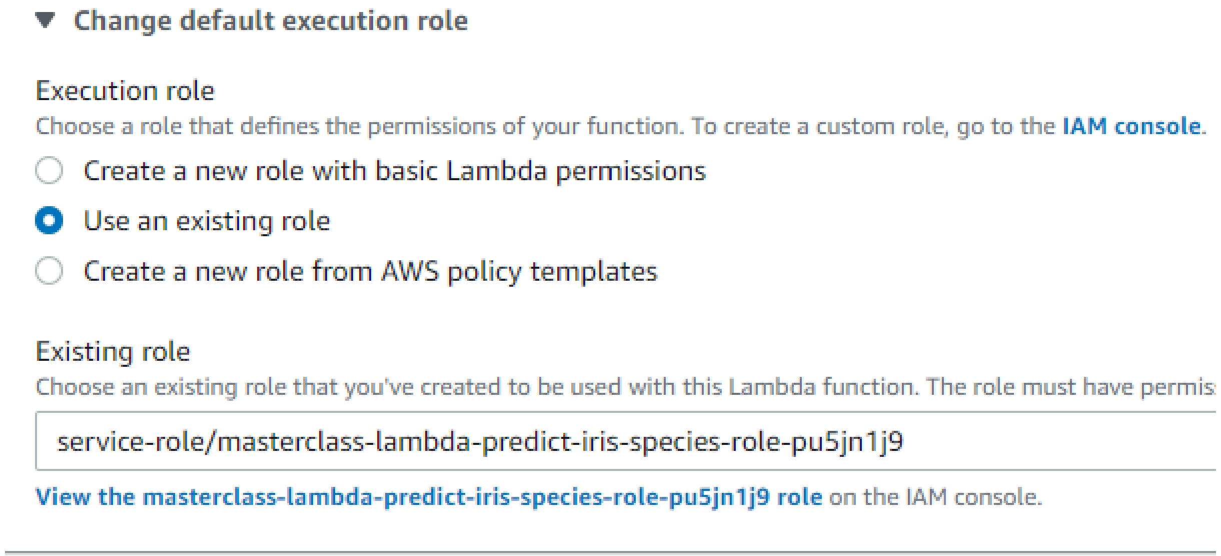
wine-model-ritesh

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the con:

Python 3.6



▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permis:

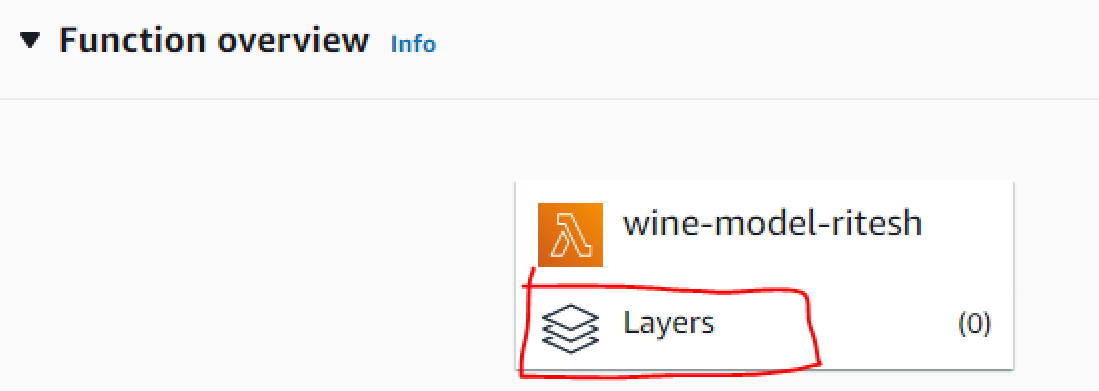
service-role/masterclass-lambda-predict-iris-species-role-pu5jn1j9

View the masterclass-lambda-predict-iris-species-role-pu5jn1j9 role on the IAM console.

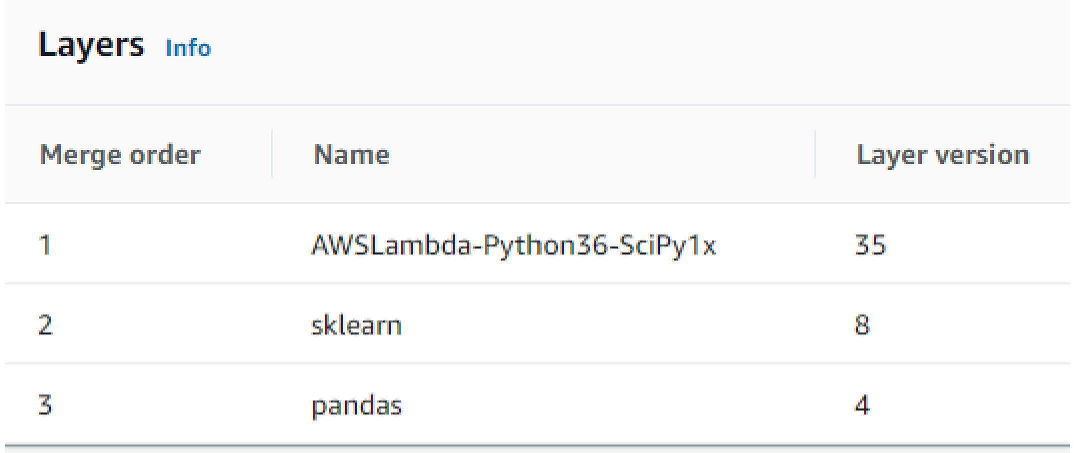
Advanced Settings - Leave as is

Adding Layers - Building lambda function

Layers provide supporting libraries.



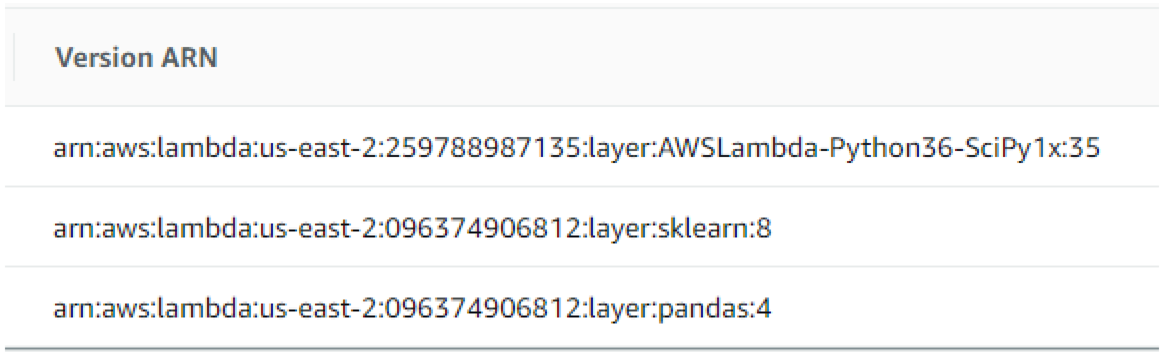
Add following layers:



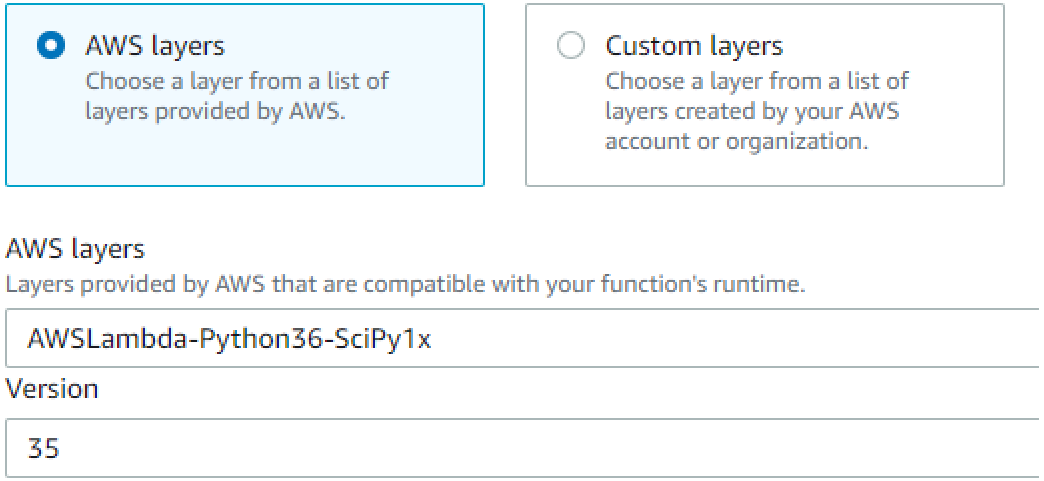
Layers Info

Merge order	Name	Layer version
1	AWSLambda-Python36-SciPy1x	35
2	sklearn	8
3	pandas	4

Building lambda function



Version ARN
arn:aws:lambda:us-east-2:259788987135:layer:AWSLambda-Python36-SciPy1x:35
arn:aws:lambda:us-east-2:096374906812:layer:sklearn:8
arn:aws:lambda:us-east-2:096374906812:layer:pandas:4



● AWS layers
Choose a layer from a list of layers provided by AWS.

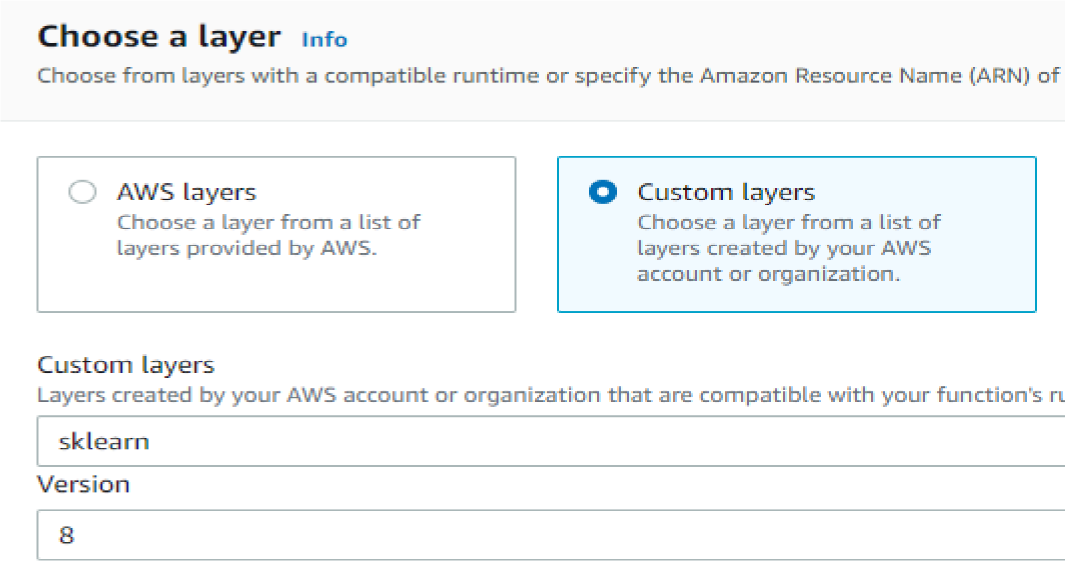
○ Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

AWS layers
Layers provided by AWS that are compatible with your function's runtime.

AWSLambda-Python36-SciPy1x

Version

35



Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of

○ AWS layers
Choose a layer from a list of layers provided by AWS.

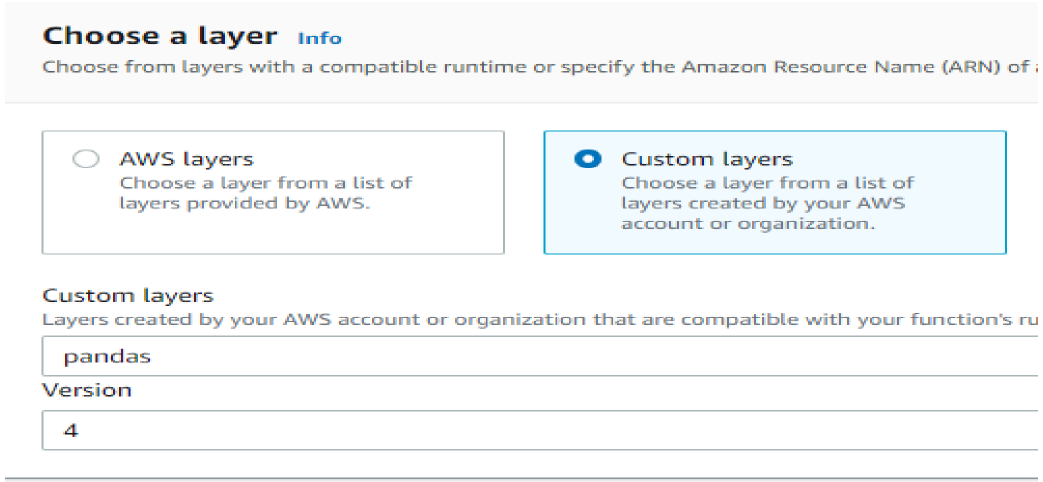
● Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

Custom layers
Layers created by your AWS account or organization that are compatible with your function's ru

sklearn

Version

8



Choose a layer Info

Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of

○ AWS layers
Choose a layer from a list of layers provided by AWS.

● Custom layers
Choose a layer from a list of layers created by your AWS account or organization.

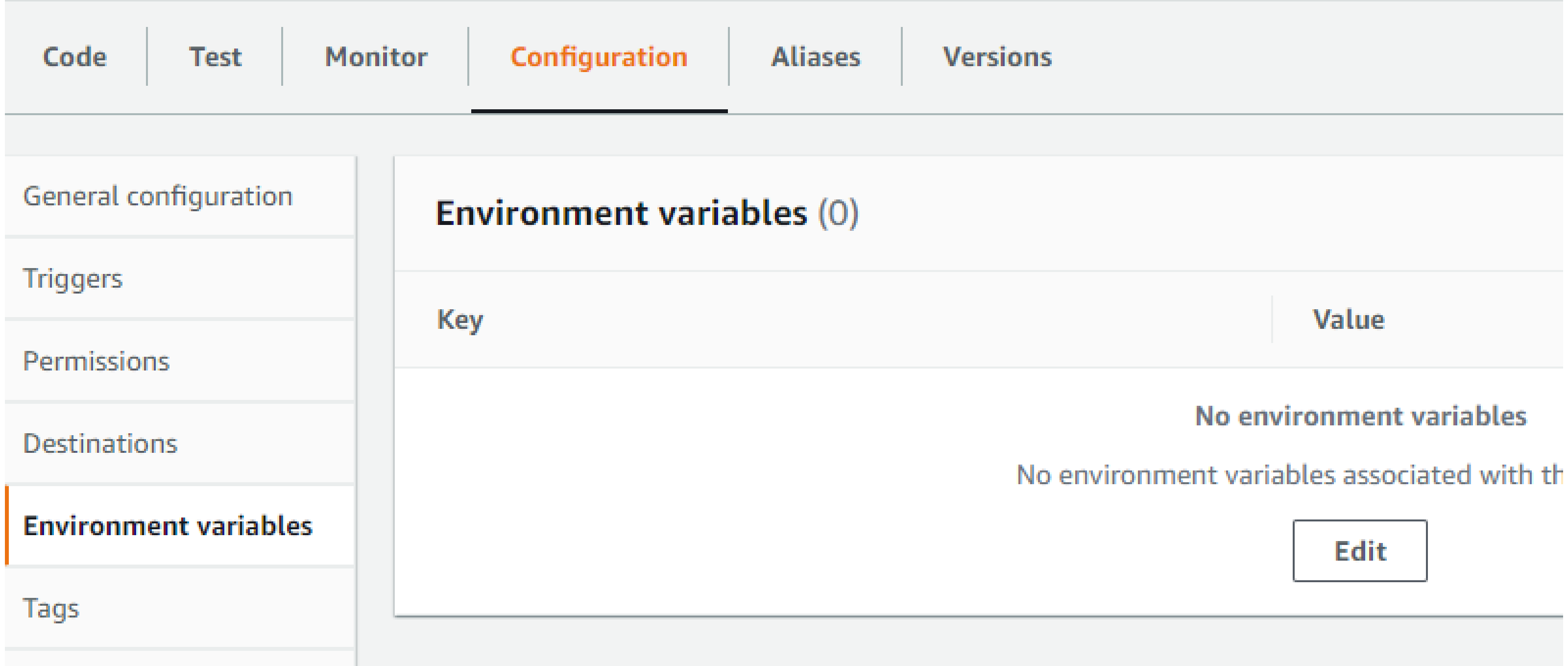
Custom layers
Layers created by your AWS account or organization that are compatible with your function's ru

pandas

Version

4

Define environment variables

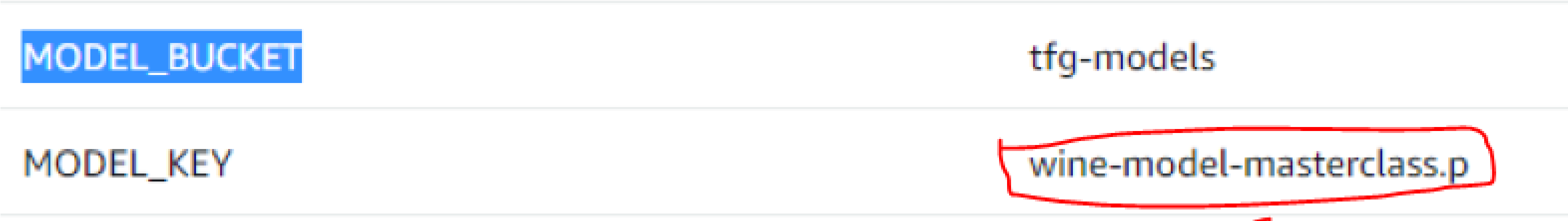


Environment variables (0)

Key	Value

No environment variables

No environment variables associated with th

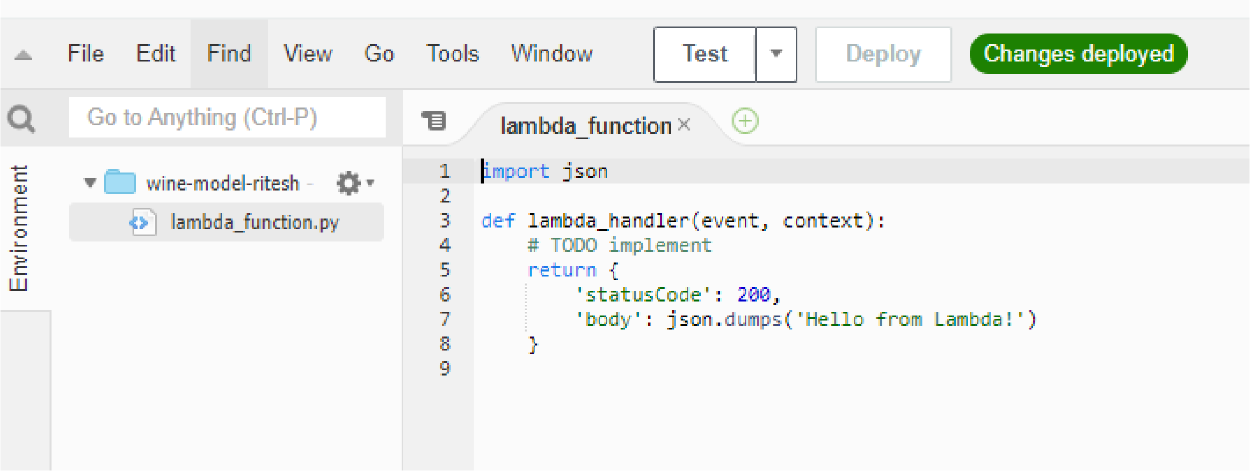


MODEL_BUCKET	tfg-models
MODEL_KEY	wine-model-masterclass.p

change

Building lambda function

IMPLEMENT CODE: click on lambda_function.py



```python
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Building lambda function

Step 1: Test that various parameters and libraries are being imported properly:

```
import json
import sklearn
import boto3
import os
import json
import pickle
import pandas as pd
```

```
def lambda_handler(event, context):
    message = f'{MODEL_BUCKET}-{MODEL_KEY}-{sklearn.__version__}'
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps(message)
    }
```

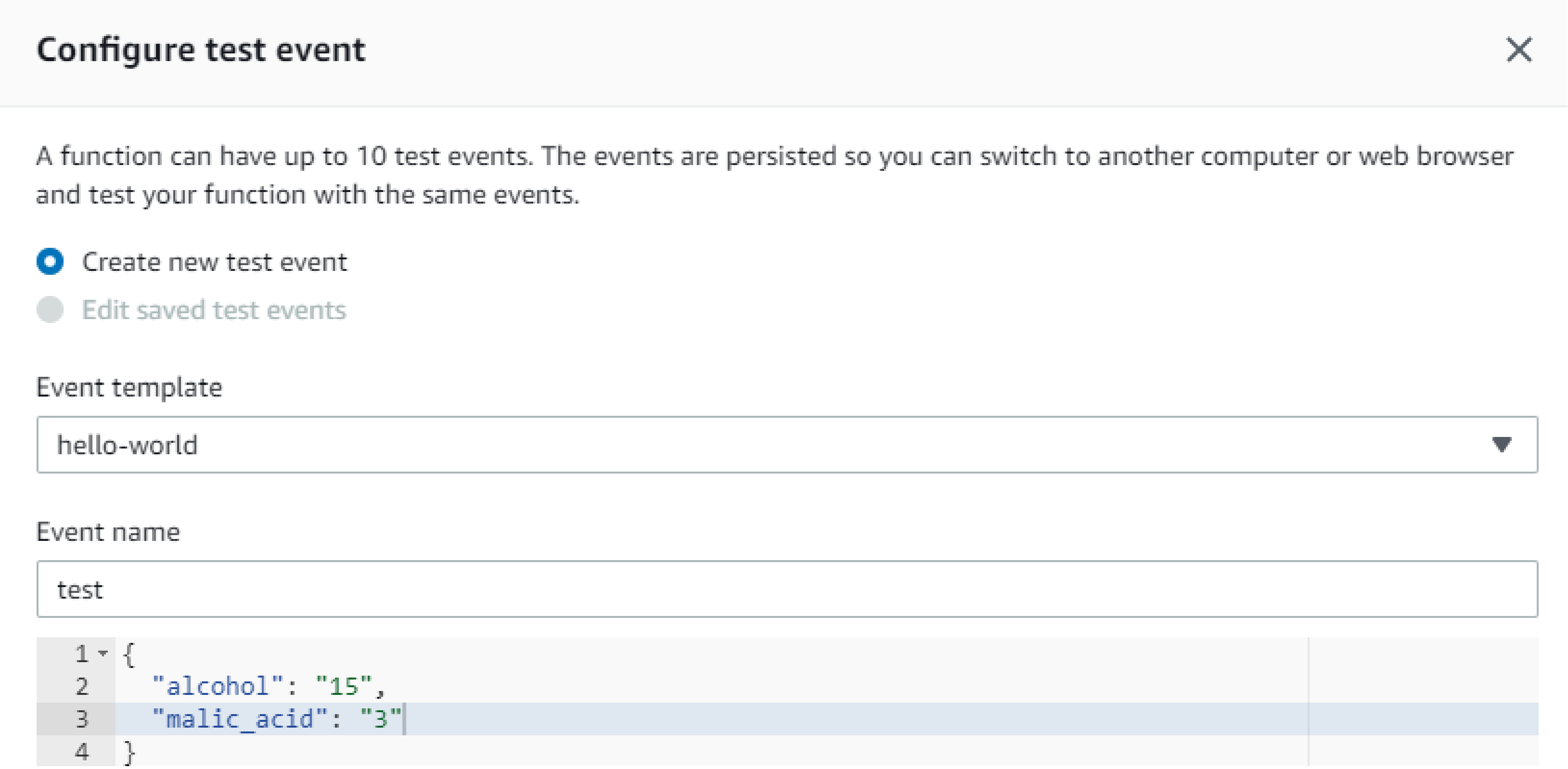
```
s3 = boto3.client('s3')
MODEL_BUCKET=os.environ.get('MODEL_BUCKET')
# get bucket prefix from ENV variable
MODEL_KEY=os.environ.get('MODEL_KEY')
```

OUTPUT:

Should be something like following:

```
{
  "statusCode": 200,
  "body": "\"tfg-models-wine-model-ritesh.p-0.24.1\""
}
```

Configure Test Event



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

● Create new test event

○ Edit saved test events

Event template

hello-world

Event name

test

```
{
  "alcohol": "15",
  "malic_acid": "3"
}
```

Loading the model - Building lambda function

MAKE THE FOLLOWING CHANGES TO LOAD THE MODEL

```
s3 = boto3.client('s3')

MODEL_BUCKET=os.environ.get('MODEL_BUCKET')

# get bucket prefix from ENV variable
MODEL_KEY=os.environ.get('MODEL_KEY')

temp_file_path = '/tmp/' + MODEL_KEY

# ** Model Init **
s3.download_file(MODEL_BUCKET, MODEL_KEY, temp_file_path)

print(temp_file_path)

with open(temp_file_path, 'rb') as f:
    model = pickle.load(f)    strclasses = str(model.classes_)
```

```
def lambda_handler(event, context):
    message = f'{MODEL_BUCKET}-{MODEL_KEY}-{strclasses}'
    return {
        'statusCode': 200,
        'body': json.dumps(message)
    }
```

OUTPUT:

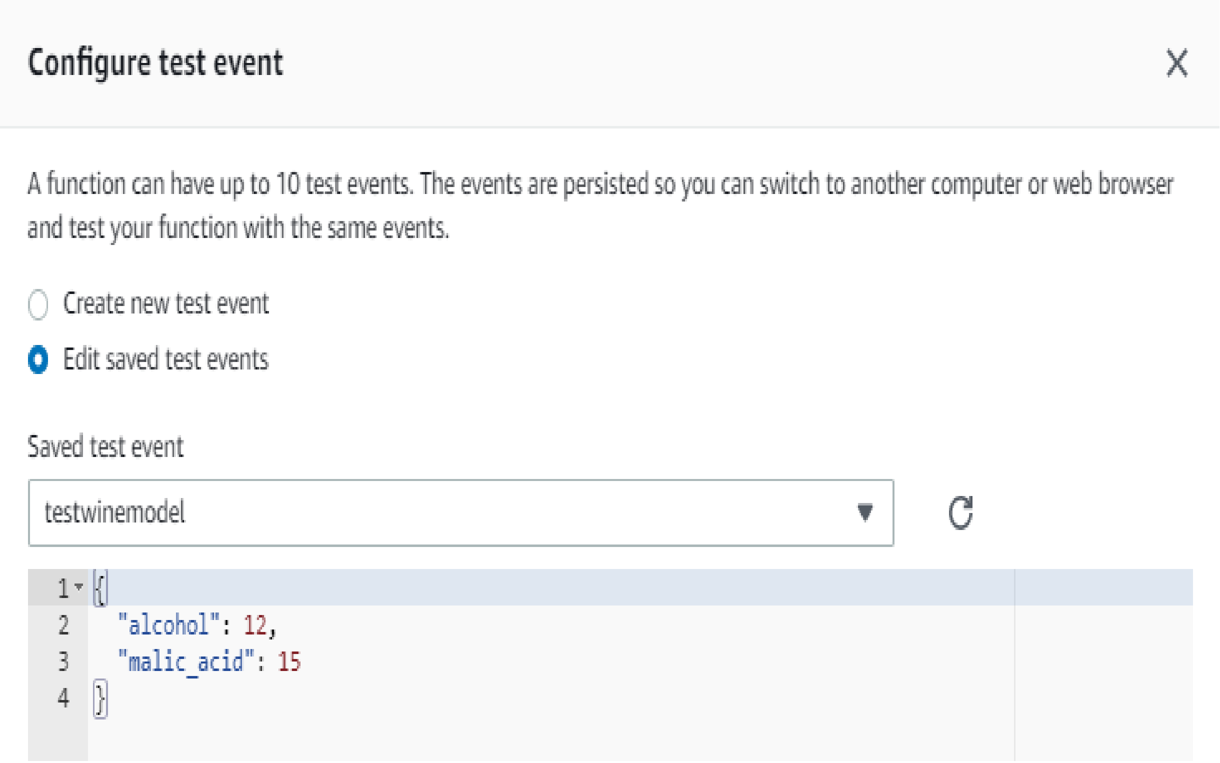
```
Response
{
  "statusCode": 200,
  "body": "\"tfg-models-wine-model-ritesh.p-[0 1 2]\""
}
```

Prediction - Building lambda function

PREDICTION - Capture Input and Call Predict Function

Inside the lambda handler module:

```
input_ = event
y_pred=model.predict(pd.DataFrame([input_]))
message = f'{MODEL_BUCKET}-{MODEL_KEY}-{y_pred}'
```



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

○ Create new test event

● Edit saved test events

Saved test event

testwinemodel

```
{
  "alcohol": 12,
  "malic_acid": 15
}
```

Output:

```
Response
{
  "statusCode": 200,
  "body": "\"tfg-models-wine-model-ritesh.p-[2]\""
}
```

Building lambda function – Exercise to Participants

Change the lambda function to calculate the time taken in prediction

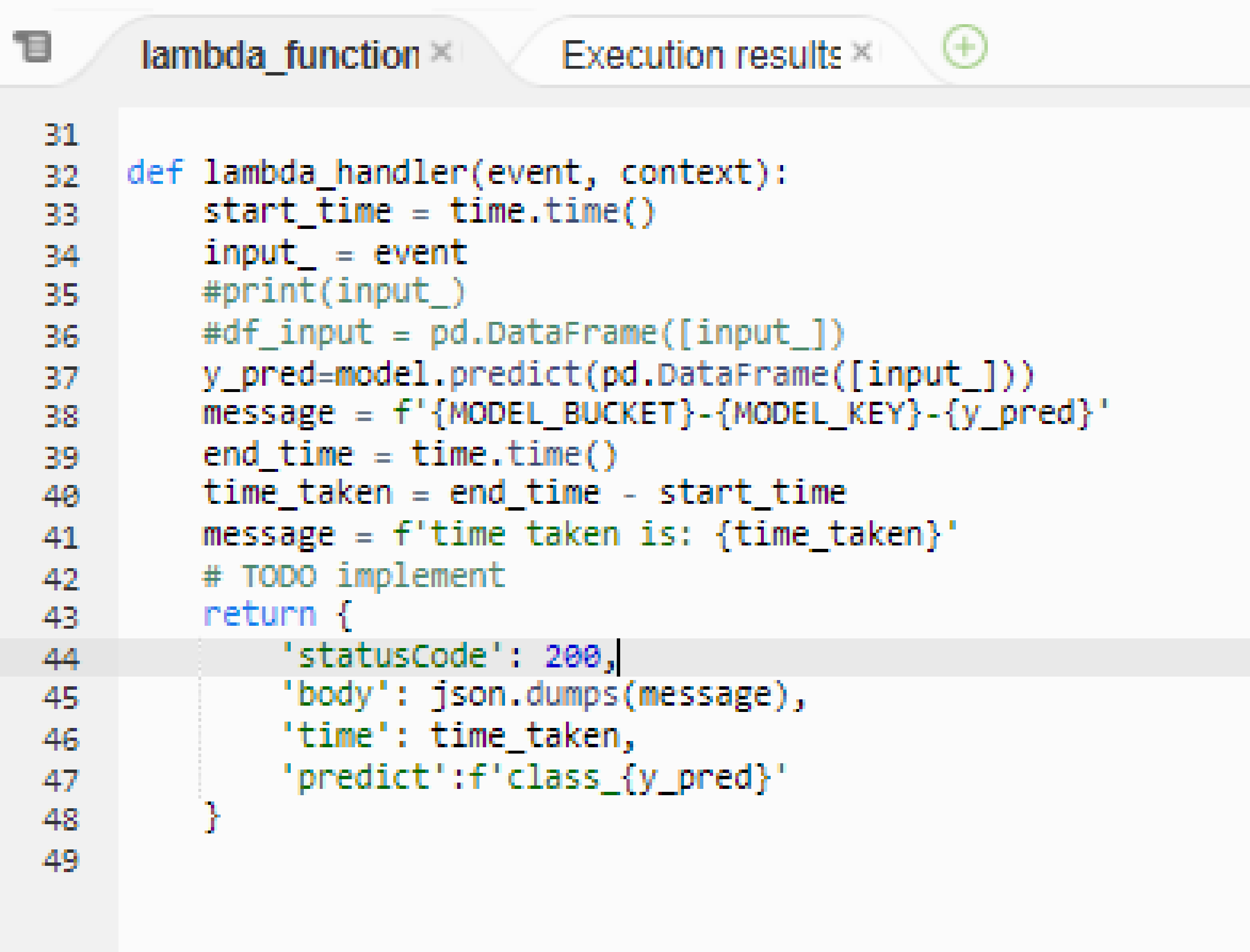
HINT: start_time = time.time()

Attach this calculated time taken to the output

Building lambda function – Exercise to Participants

SOLUTION

Building lambda function – Exercise to Participants

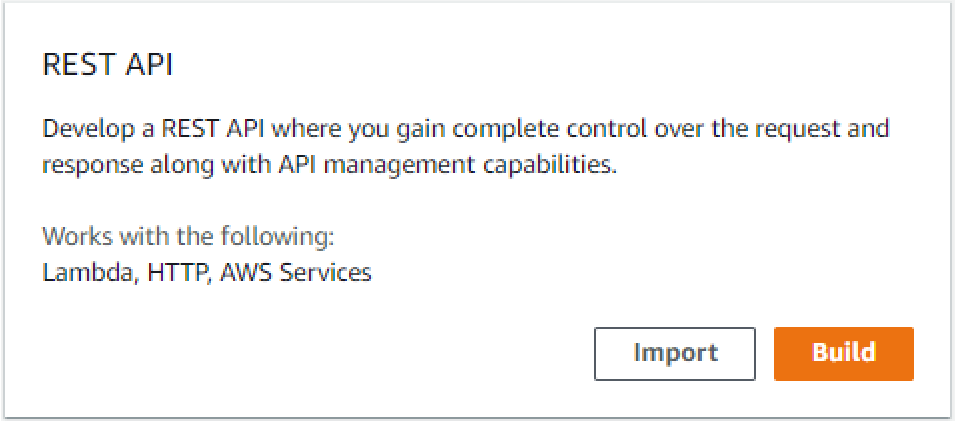


```python

def lambda_handler(event, context):
    start_time = time.time()
    input_ = event
    #print(input_)
    #df_input = pd.DataFrame([input_])
    y_pred=model.predict(pd.DataFrame([input_]))
    message = f'{MODEL_BUCKET}-{MODEL_KEY}-{y_pred}'
    end_time = time.time()
    time_taken = end_time - start_time
    message = f'time taken is: {time_taken}'
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps(message),
        'time': time_taken,
        'predict':f'class_{y_pred}'
    }

```

Integrate with API gateway

- Create API
- Create Resource
- Create Method
- Test on Platform
- Test with Curl

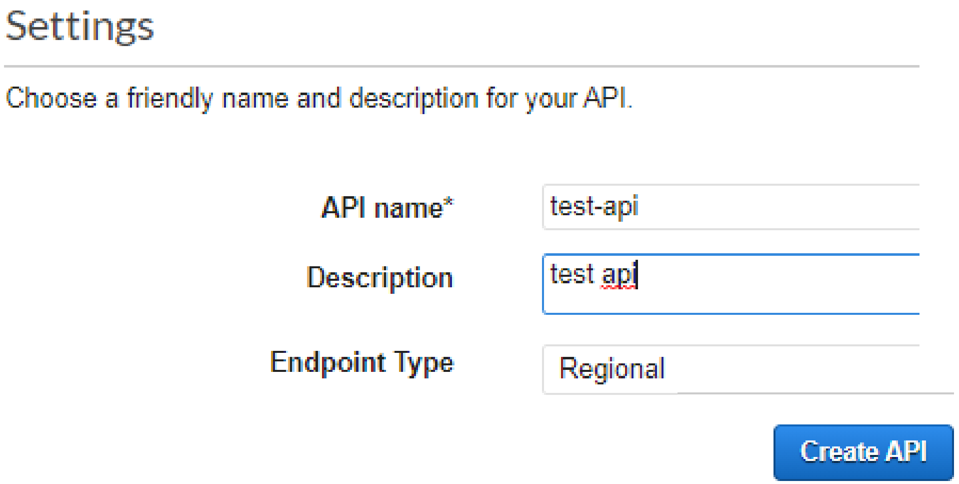
Create API



REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

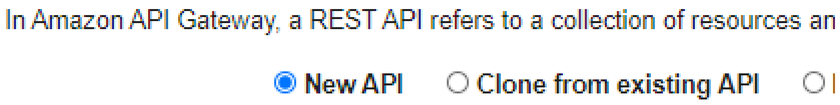
Works with the following:
Lambda, HTTP, AWS Services



Settings

Choose a friendly name and description for your API.

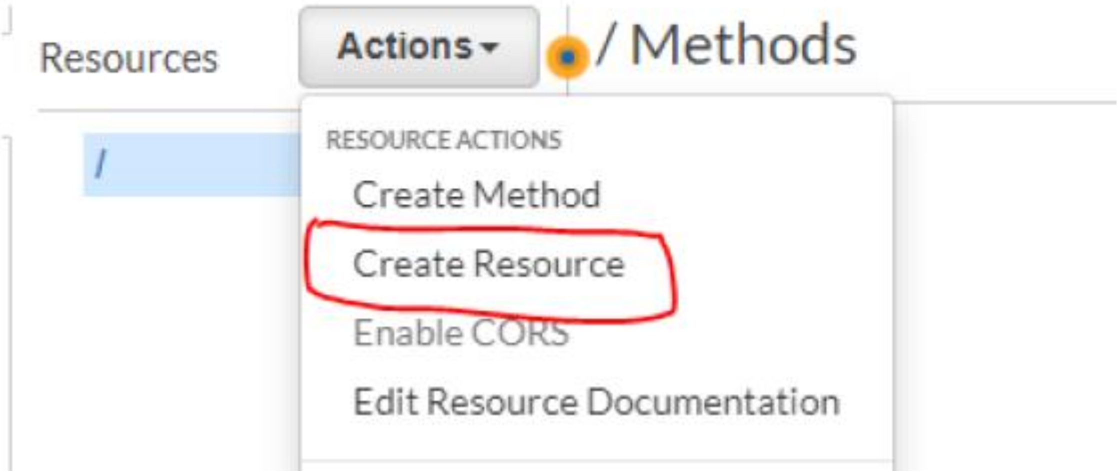
API name*	test-api
Description	test api
Endpoint Type	Regional

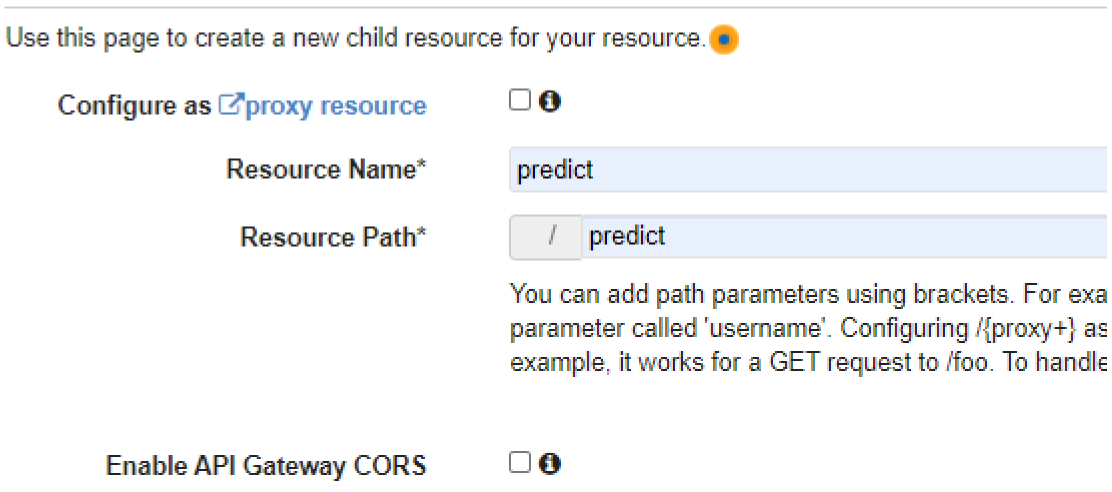


In Amazon API Gateway, a REST API refers to a collection of resources an

● New API ○ Clone from existing API ○ I

Create Resource





Use this page to create a new child resource for your resource.

Configure as proxy resource □

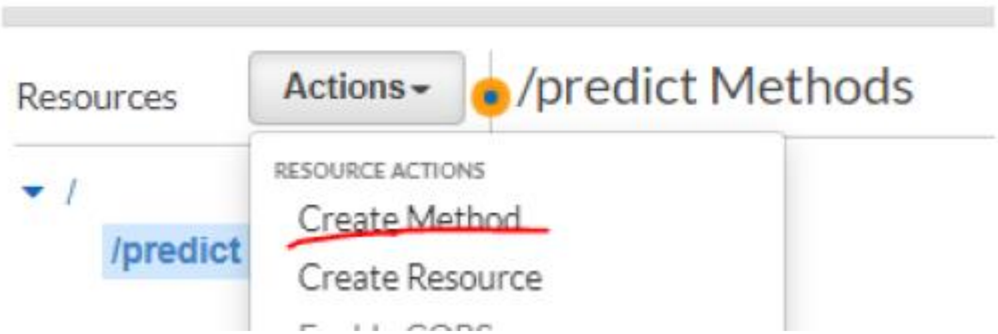
Resource Name* predict

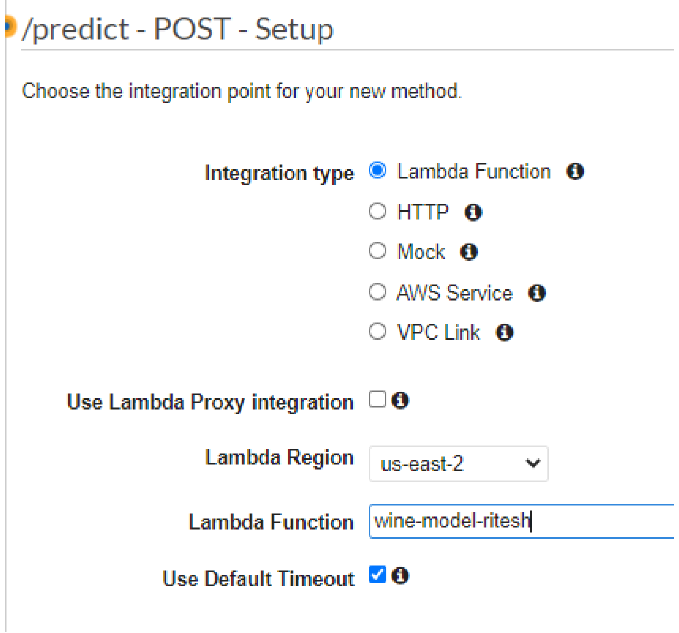
Resource Path* / predict

You can add path parameters using brackets. For exa
parameter called 'username'. Configuring /{proxy+} as
example, it works for a GET request to /foo. To handle

Enable API Gateway CORS □

Create Method




/predict - POST - Setup

Choose the integration point for your new method.

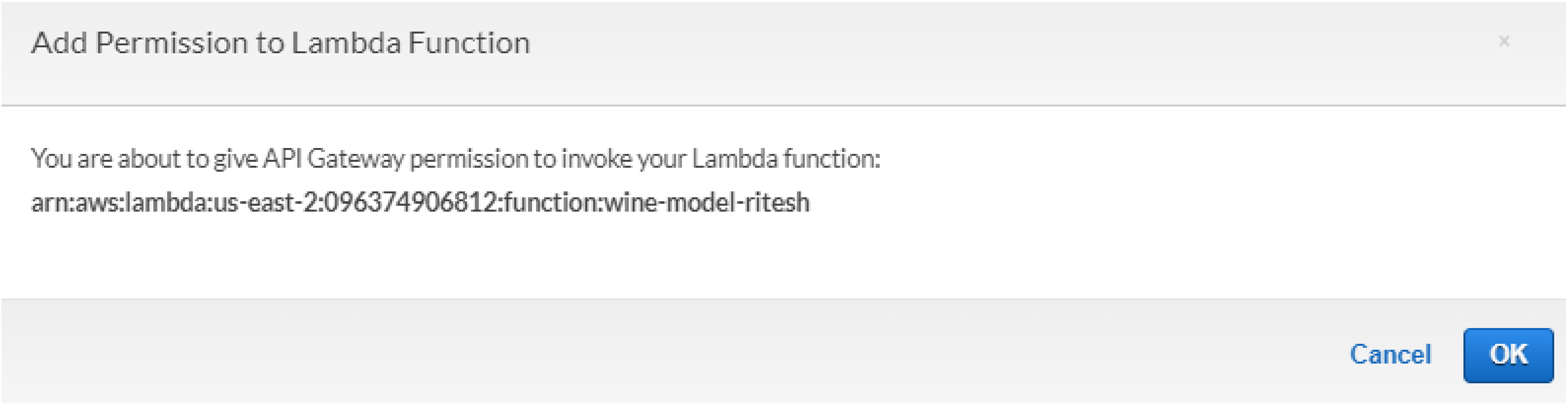
Integration type ● Lambda Function ○ HTTP ○ Mock ○ AWS Service ○ VPC Link

Use Lambda Proxy integration □

Lambda Region us-east-2

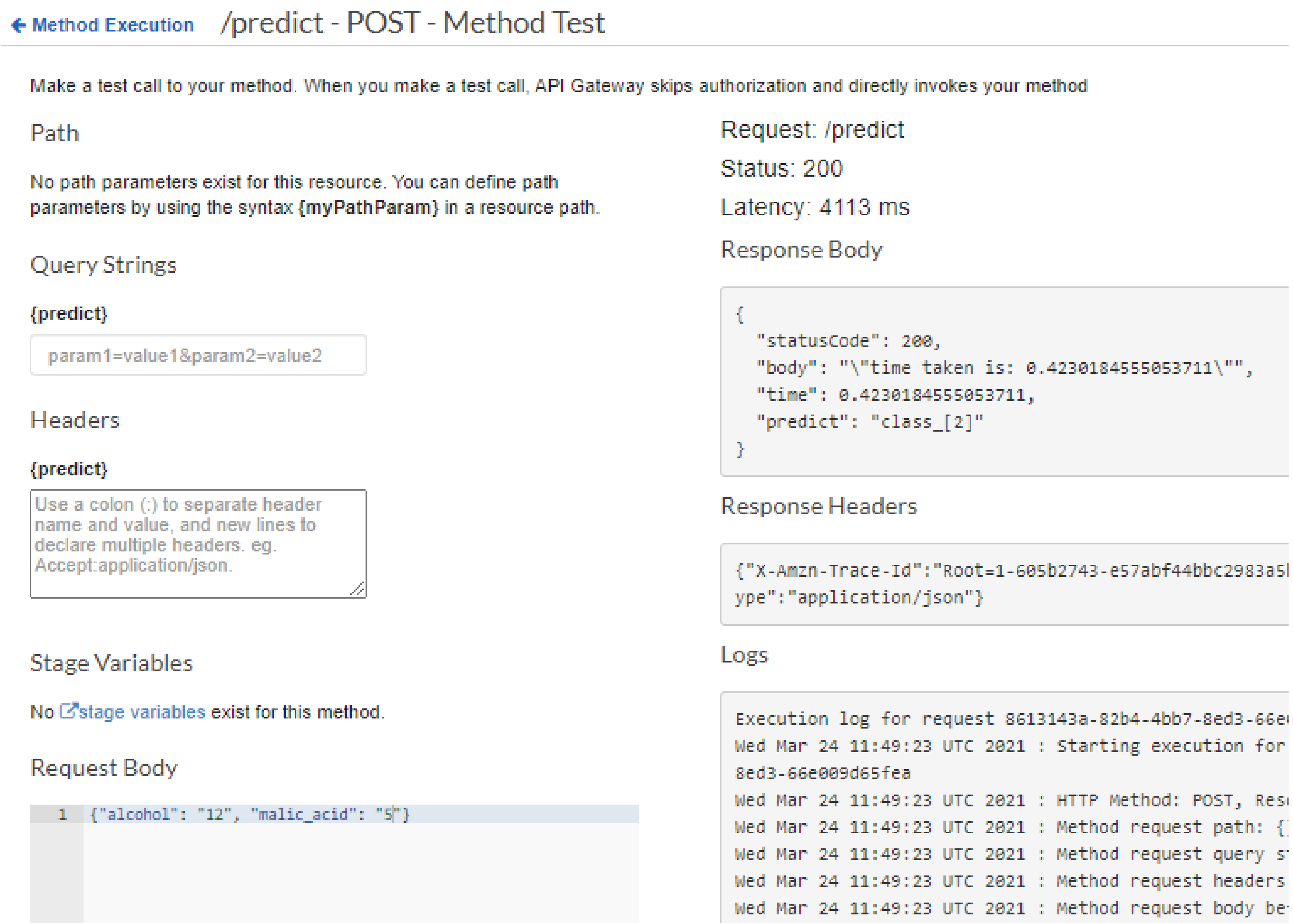
Lambda Function wine-model-ritesh

Use Default Timeout ☑


Add Permission to Lambda Function

You are about to give API Gateway permission to invoke your Lambda function:
arn:aws:lambda:us-east-2:096374906812:function:wine-model-ritesh

Test Method



/predict - POST - Method Test

Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method

Path

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

Query Strings

{predict}

param1=value1&param2=value2

Headers

{predict}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.

Stage Variables

No stage variables exist for this method.

Request Body

```
{"alcohol": "12", "malic_acid": "5"}
```

Request: /predict

Status: 200

Latency: 4113 ms

Response Body

```
{
  "statusCode": 200,
  "body": "\"time taken is: 0.4230184555053711\"",
  "time": 0.4230184555053711,
  "predict": "class_[2]"
}
```

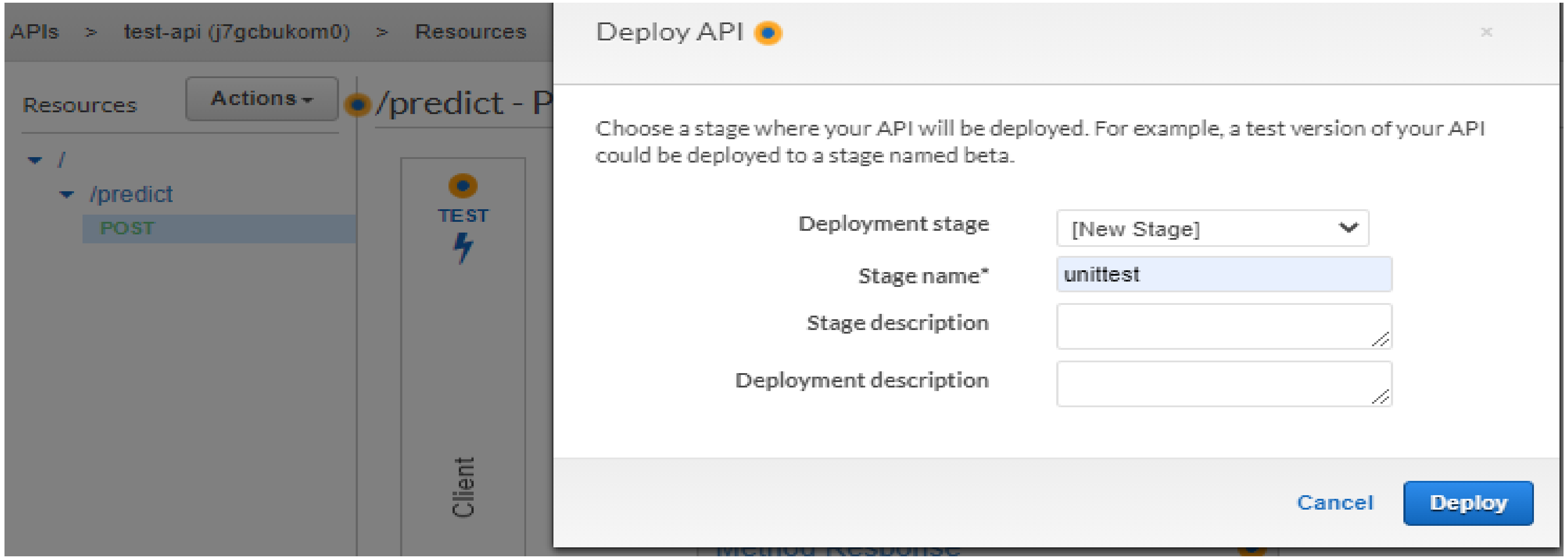
Response Headers

```
{"X-Amzn-Trace-Id":"Root=1-605b2743-e57abf44bbc2983a5
ype":"application/json"}
```

Logs

```
Execution log for request 8613143a-82b4-4bb7-8ed3-66e
Wed Mar 24 11:49:23 UTC 2021 : Starting execution for
8ed3-66e009d65fea
Wed Mar 24 11:49:23 UTC 2021 : HTTP Method: POST, Res
Wed Mar 24 11:49:23 UTC 2021 : Method request path: {
Wed Mar 24 11:49:23 UTC 2021 : Method request query s
Wed Mar 24 11:49:23 UTC 2021 : Method request headers
Wed Mar 24 11:49:23 UTC 2021 : Method request body be
```

Deploy API



Deploy API

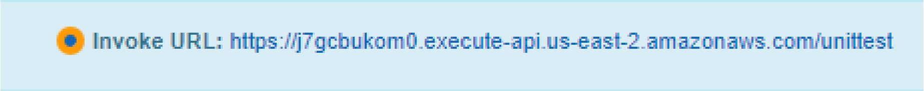
Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage: [New Stage]

Stage name*: unittest

Stage description:

Deployment description:

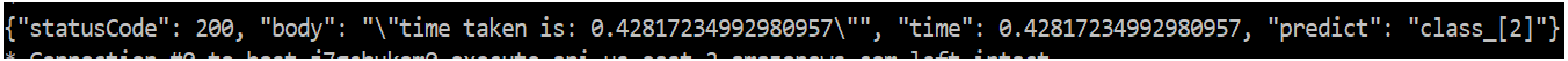


Invoke URL: https://j7gcbukom0.execute-api.us-east-2.amazonaws.com/unittest

COPY THE URL : https://j7gcbukom0.execute-api.us-east-2.amazonaws.com/unittest

Test from command line

```
curl -v -X POST "https://<subdomain>.execute-api.us-east-2.amazonaws.com/unittest/predict/" -H "content-type: application/json" -d "{ \"alcohol\": \"12\", \"malic_acid\": \"15\" }"
```



```
{"statusCode": 200, "body": "\"time taken is: 0.42817234992980957\"", "time": 0.42817234992980957, "predict": "class_[2]"}
```

Discussion

Creating a layer:

```
import os
import shutil
os.listdir('.')
! pip3 install --ignore-installed --target=python pandas
# ! pip3 install --ignore-installed --target=python scikit-learn ‘ for scikit learn
! rm -rf python/numpy* python/scipy*
! zip -r ./pandas.zip python
```

Deploying a deep NN model

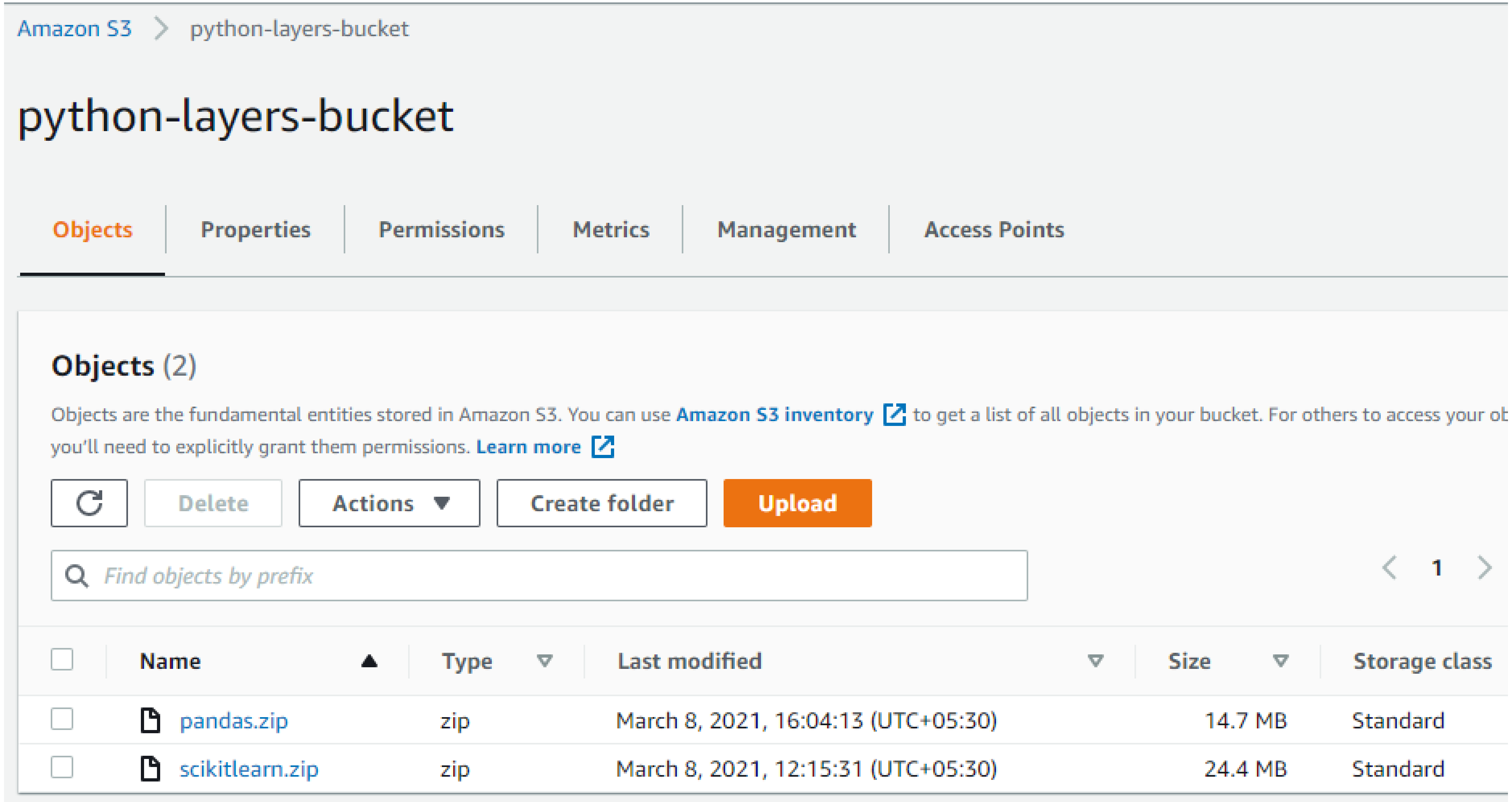
https://github.com/sinharitesh/lambda-multilabel

Exercise for participants

Advanced Exercise:
Move the code loading to lambda handler call and note the time taken. is there a drop in subsequent calls. that will tell if there is time difference.

DELETED

BACKUP



Amazon S3 > python-layers-bucket

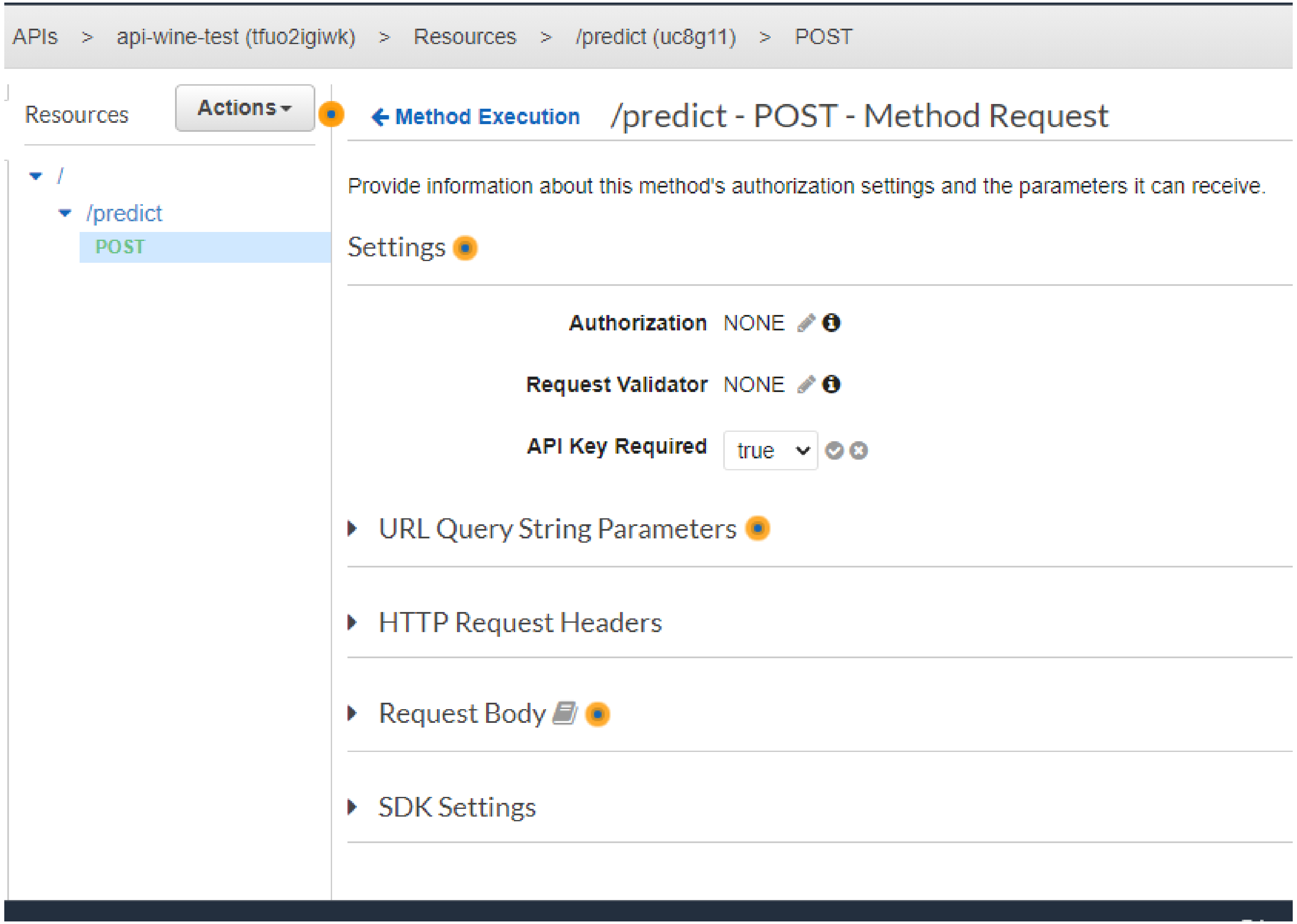
python-layers-bucket

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use **Amazon S3 inventory** to get a list of all objects in your bucket. For others to access your ob you'll need to explicitly grant them permissions. **Learn more**

□	Name	Type	Last modified	Size	Storage class
□	pandas.zip	zip	March 8, 2021, 16:04:13 (UTC+05:30)	14.7 MB	Standard
□	scikitlearn.zip	zip	March 8, 2021, 12:15:31 (UTC+05:30)	24.4 MB	Standard

Securing the API



APIs > api-wine-test (tfuo2igiwk) > Resources > /predict (uc8g11) > POST

/predict - POST - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization NONE

Request Validator NONE

API Key Required true

- URL Query String Parameters
- HTTP Request Headers
- Request Body
- SDK Settings