Session 1 - Deploying over AWS

Due 22 Jul 2020 by 23:59 **Points** 1,000 **Submitting** a website url
Available 11 Jul 2020 at 9:45 - 22 Jul 2020 at 23:59 12 days

This assignment was locked 22 Jul 2020 at 23:59.

Session 1 - Deploying over AWS

SERVERLESS COMPUTING

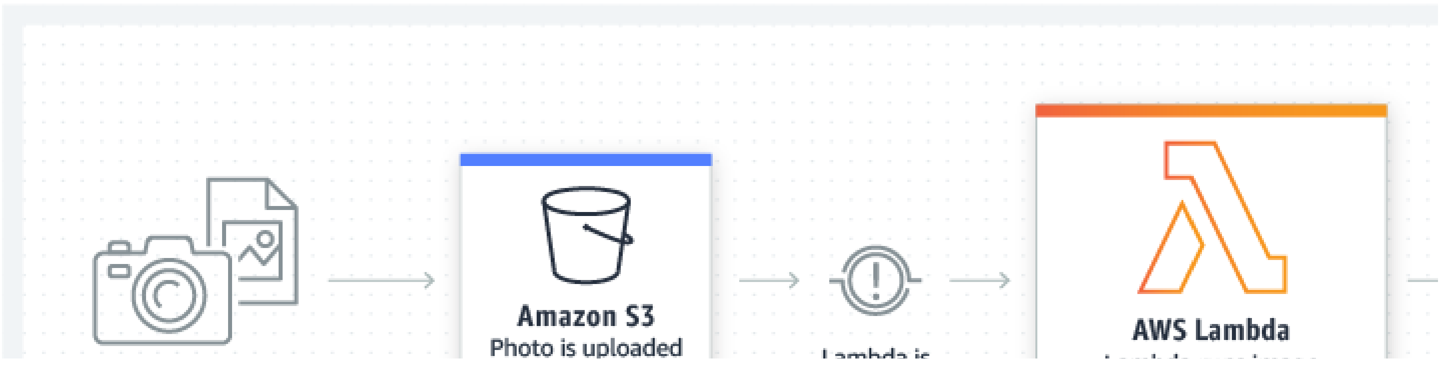
SERVERLESS COMPUTING

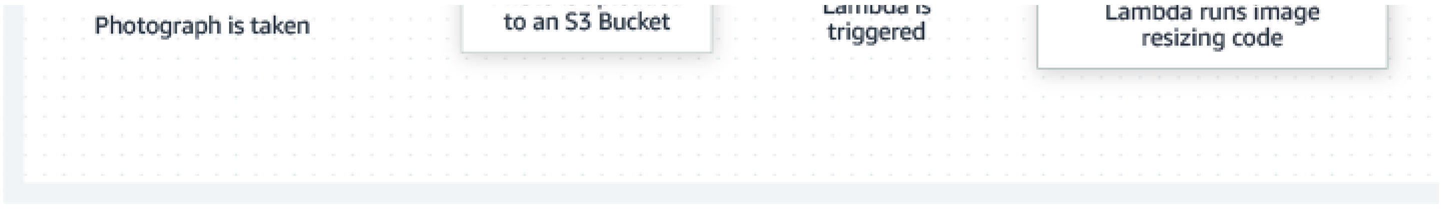
Build and run applications without thinking about servers

- Serverless allows you to build and run applications and services without thinking about servers.
- Cloud-based computing model.
- It eliminates **(hides)** infrastructure management tasks such as:
  - server or cluster provisioning
  - operating system maintenance
  - capacity provisioning
  - downtime
- The unit of execution is often a function:
  - FaaS - Function as a Service
- Pricing is based on the actual amount of resources consumed
- You can build them for nearly any type of application or backend service, and **everything required to run and scale your application with high availability is handled for you!**

AWS LAMBDA

- AWS Lambda lets you run code without provisioning or managing servers.
- You pay only for the compute time you consume - there is no charge when your code is not running
- It automatically scales your application by running code in response to each trigger. Your code runs in parallel and process each trigger individually, scaling precisely with the size of the workload
- You are charged for every 100ms your code executes
- Can be triggered by different events (e.g. HTTP request)
- Your function code and it's dependencies shouldn't be greater than 250MB (deployment package .zip file)



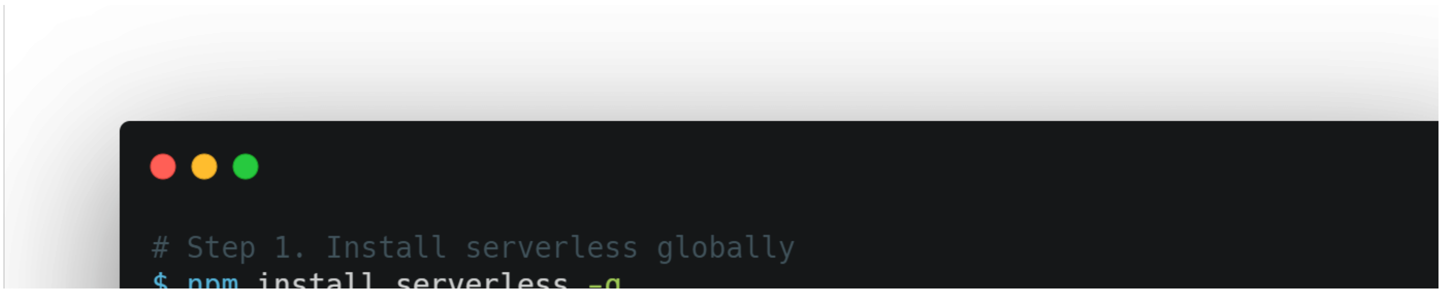


- Runs on Amazon Linux OS, inside a runtime container
- The more RAM memory you allocate, the faster CPU you get:
  - from 128MB to 3008 MB, in 64 increments
- Max. 500MB of storage inside /tmp directory (where your code/files can expand)
- Free Tier:
  - 1 million requests per month ($0.0000002 per requests thereafter)
  - 400,000 GB-seconds of compute time per month ($0.000000208 for every 100ms used thereafter, 128 MB of RAM)

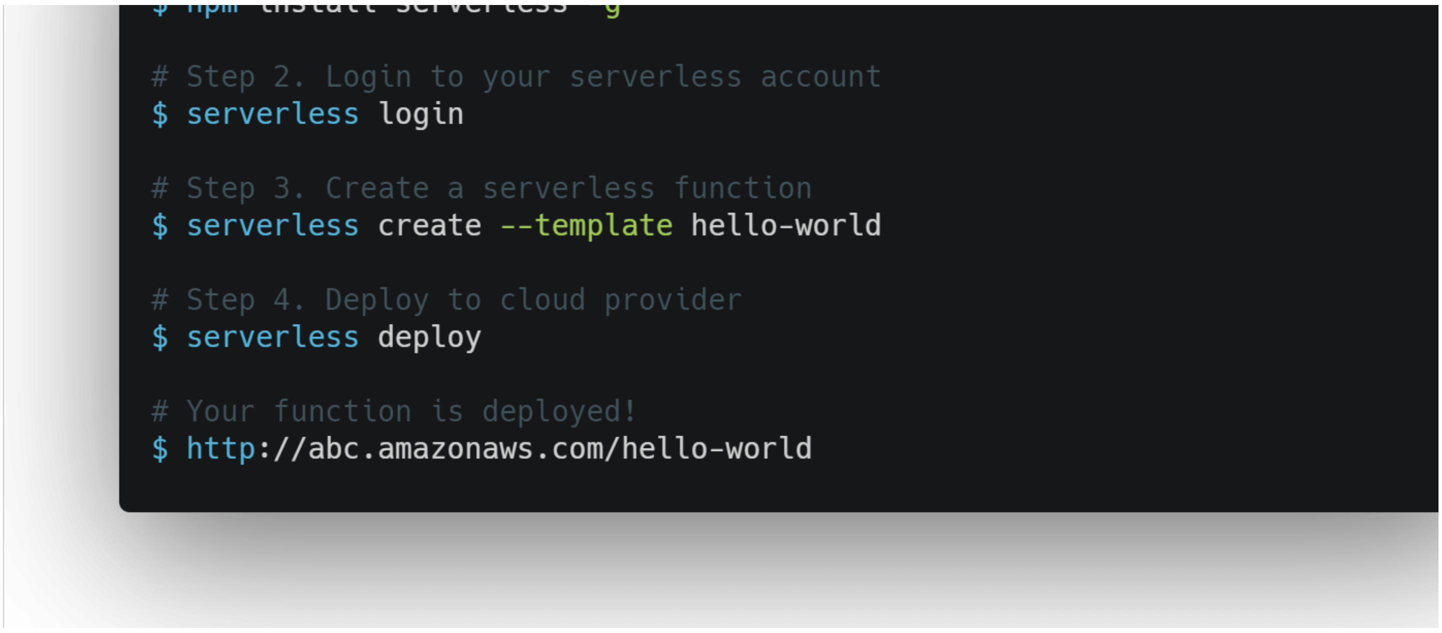
WHAT IS SERVERLESS FRAMEWORK?

The **Serverless Framework** is a free and **open-source (https://en.wikipedia.org/wiki/Open_Source) web framework (https://en.wikipedia.org/wiki/Web_framework)** written using **Node.js (https://en.wikipedia.org/wiki/Node.js)**. Serverless is the first framework developed for building applications on **AWS Lambda (https://en.wikipedia.org/wiki/Amazon_Lambda)**, a **serverless computing (https://en.wikipedia.org/wiki/Serverless_computing)** platform provided by **Amazon (https://en.wikipedia.org/wiki/Amazon.com)** as a part of **Amazon Web Services (https://en.wikipedia.org/wiki/Amazon_Web_Services)** but now can work with others as well.

- A Serverless app can simply be a couple of lambda functions to accomplish some tasks or an entire back-end composed of hundreds of lambda functions.
- It is used for building serverless applications via command line and config files
- It uses Amazon CloudFormation under the hood, which allows you to describe and provision all the infrastructural resources you need using a single JSON file. Once we define this file, all resources will be created in the cloud for us.
- Easy and fun!

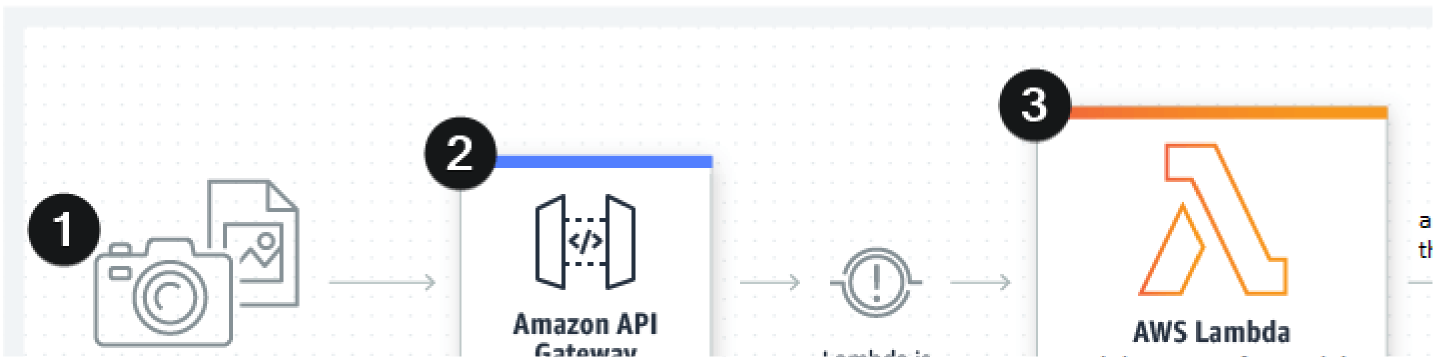


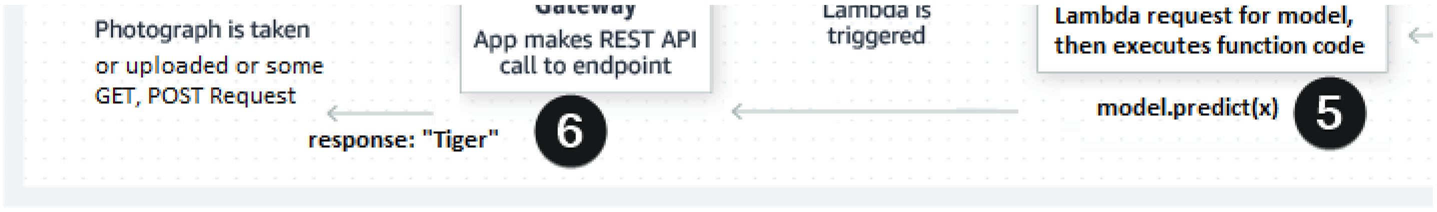
```
# Step 1. Install serverless globally
$ npm install serverless -g
```



```
$ npm install serverless -g

# Step 2. Login to your serverless account
$ serverless login

# Step 3. Create a serverless function
$ serverless create --template hello-world

# Step 4. Deploy to cloud provider
$ serverless deploy

# Your function is deployed!
$ http://abc.amazonaws.com/hello-world
```

WHAT EXACTLY ARE WE GOING TO DO?




Photograph is taken or uploaded or some GET, POST Request
Gateway
App makes REST API call to endpoint
Lambda is triggered
Lambda request for model, then executes function code
response: "Tiger"
6
model.predict(x)
5

Let's take a quick **QUIZ1 (https://s.surveyplanet.com/G2STy8at8r)** now! 😎

LINUX

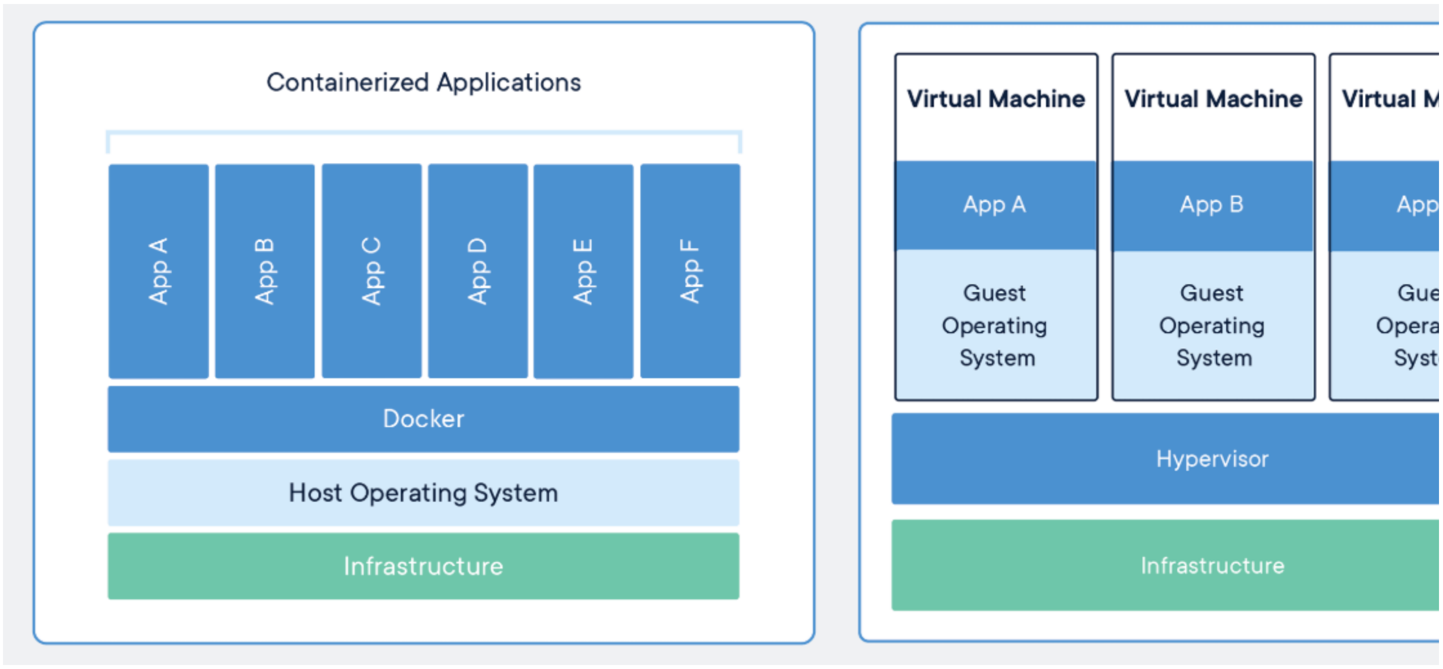
- Setup is faster
- Everyone will have the same issues 😅
- You can use all features from serverless-python-requirements plugin
- You need Windows 10 Pro or Enterprise for the Docker we need.
- You can:
  - use your laptop if you already have Linux
  - use VirtualBox and install Ubuntu on it
  - use macOS (with some online help you should be good).

WHAT IS DOCKER?

- Docker is a tool for creating, deploying and running applications inside containers
- Containers are standardized software units that contain code, dependencies, runtimes, and settings in a single package
- Containers store applications by keeping them isolated from the environment they run on
- They are portable and ensure that application works uniformly on different computing environments
- Container images become containers when they run on Docker Engine (shown as App below in left image)

left image)

- Containers virtualize the OS components instead of hardware whereas VMs abstract physical hardware
- Each container is an isolated process, but multiple containers can share the same OS kernel with others whereas, Each VM includes a full copy of an OS, apps, and libs
- Containers usually take up less space (in tens of MBs) and require fewer resources, whereas, VMs can take up tens of GBs and can be slower.



Docker in the context of Serverless Framework

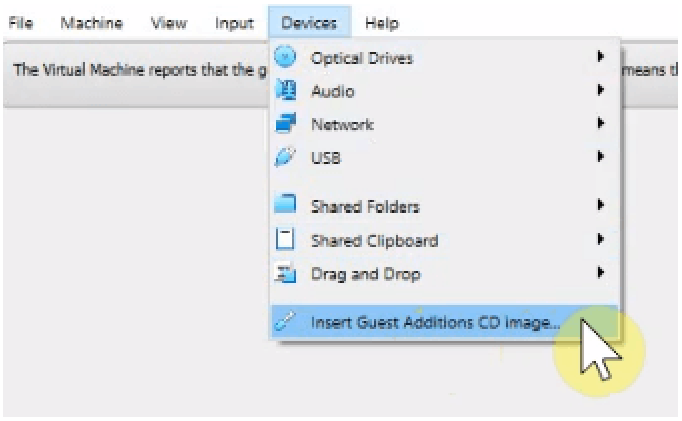
- Framework plugins can use Docker when installing Python requirements using pip
- Used Docker container is similar to the Lambda environment so the compiled dependencies will be compatible
- Dependencies will be installed using containers and then zipped to the Lambda deployment package.

BORING VERY IMPORTANT STEPS

CONFIGURING UBUNTU | NODEJS | SERVERLESS

1. Install **VirtualBox** **(https://www.virtualbox.org/wiki/Downloads)** Your password will be osboxes.org
2. Download **Ubuntu 20.04 from Osboxes.org** **(https://www.osboxes.org/ubuntu/)**
3. Create the best possible Virtual Machine on Virtual Box with a minimum

1. 4GB+ RAM and then
2. Select Use an existing virtual hard disk file and then point to your extracted Ubuntu 20.04.vdi
3. Provide as many CPU cores as possible
4. Your resolution wouldn't be great. Install Guest Editions CD images (Google in case you're still not able to fix it)



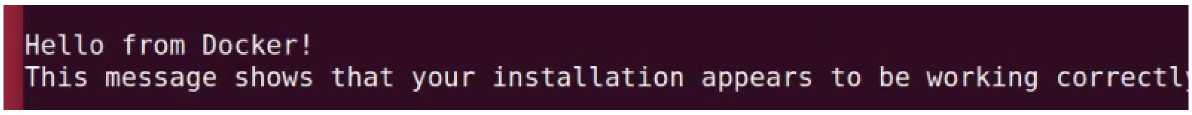
The Virtual Machine reports that the g … means t

5. In Ubuntu >> Settings >> Power >> Disable Power savings:
   1. Power Saving - Blank Screen - Never
   2. Automatic Suspend - On Battery Power - OFF
6. Create a Snapshot of your Virtual Machine. In case anything goes bad later on, you can always use this snapshot.
7. Install Miniconda **https://docs.conda.io/en/latest/miniconda.html** **(https://docs.conda.io/en/latest/miniconda.html)** then Miniconda3 Linux 64-Bit
   1. **https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh** **(https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh)**
8. Open Terminal and navigate to the Downloads folder, and **bash Miniconda3-latest-linux-x86_64.sh** (the sh file we downloaded in the last step)
9. Use Ubuntu Sofware to install the Visual Studio Code (or any other code editor of your choice). Install Python Extension in VS Code.
10. Install **pip** Run these commands in order:
    1. **sudo apt-get install python3-distutils**
    2. **wget http://bootstrap.pypa.io/get-pip.py** **(http://bootstrap.pypa.io/get-pip.py)**
    3. **sudo python get-pip.py**
       1. if you get this error **"launchpadlib x.x.x requires testresources, which is not installed"** then run
          sudo pip3 install testresources
11. Install Docker:

    1. Visit: **https://download.docker.com/linux/ubuntu/dists/bionic/pool/stable/amd64/** **(https://download.docker.com/linux/ubuntu/dists/bionic/pool/stable/amd64/)**
    2. Download these 3 files (you should download latest version, below are the files I used):
       1. docker-ce_19.03.9~3-0~ubuntu-bionic_amd64.deb
       2. docker-ce-cli_19.03.9~3-0~ubuntu-bionic_amd64.deb
       3. containerd.io_1.2.13-2_amd64.deb
    3. use **sudo dpkg -i containerd.io_1.2.13-2_amd64.deb** for example to install containerd... Install all 3

install all 3

4. Make Docker to start on boot:
   1. **sudo systemctl enable docker**
5. Check if Docker is installed properly
   1. **sudo docker run hello-world**
      1. If you see below it means everything went fine:

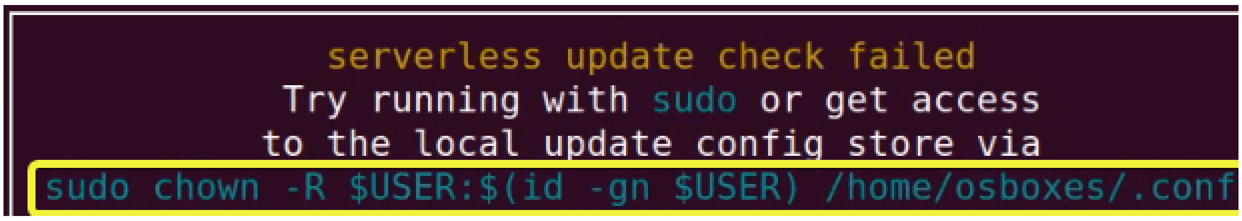


```
Hello from Docker!
This message shows that your installation appears to be working correctl
```

      2. For me, it did not. I got an error "*Got permission denied while trying to connect to the Docker daemon socket*"
         1. this helped: **sudo chmod 666 /var/run/docker.sock on**
         2. Refer to this **[link](https://www.digitalocean.com/community/questions/how-to-fix-docker-got-permission-denied-while-trying-to-connect-to-the-docker-daemon-socket)** **(https://www.digitalocean.com/community/questions/how-to-fix-docker-got-permission-denied-while-trying-to-connect-to-the-docker-daemon-socket)** in case you also see this error in the future.
         3. If you still face this problem (docker doesn't start):
            1. service docker stop
               cd /var/run/docker/libcontainerd
               rm -rf containerd/*
               rm -f docker-containerd.pid
               service docker start
         4. If everything fails, follow these steps:
            https://docs.docker.com/engine/install/ubuntu/ (https://docs.docker.com/engine/install/ubuntu/)

12. Install Node.js:
    1. Start with installing ***curl***
       sudo apt install curl
    2. Download an installation script that will add a PPA (Personal Package Archive) to your Ubuntu Package Repositories maintained by Node to get the latest updated node packages.
       curl -sL https://deb.nodesource.com/setup_10.x (https://deb.nodesource.com/setup_10.x) -o nodesource_setup.sh
       sudo bash nodesource_setup.sh
       sudo apt-get install -y nodejs
    3. Check the installation by checking **nodejs** and **npm** versions
       node -v
       npm -v

13. Install the Serverless framework:
    1. **sudo npm install -g serverless**
       1. you most probably will get this:



```
serverless update check failed
Try running with sudo or get access
to the local update config store via
sudo chown -R $USER:$(id -gn $USER) /home/osboxes/.conf
```

2. just copy the command in yellow: "**sudo chown -R...** " command and execute it.

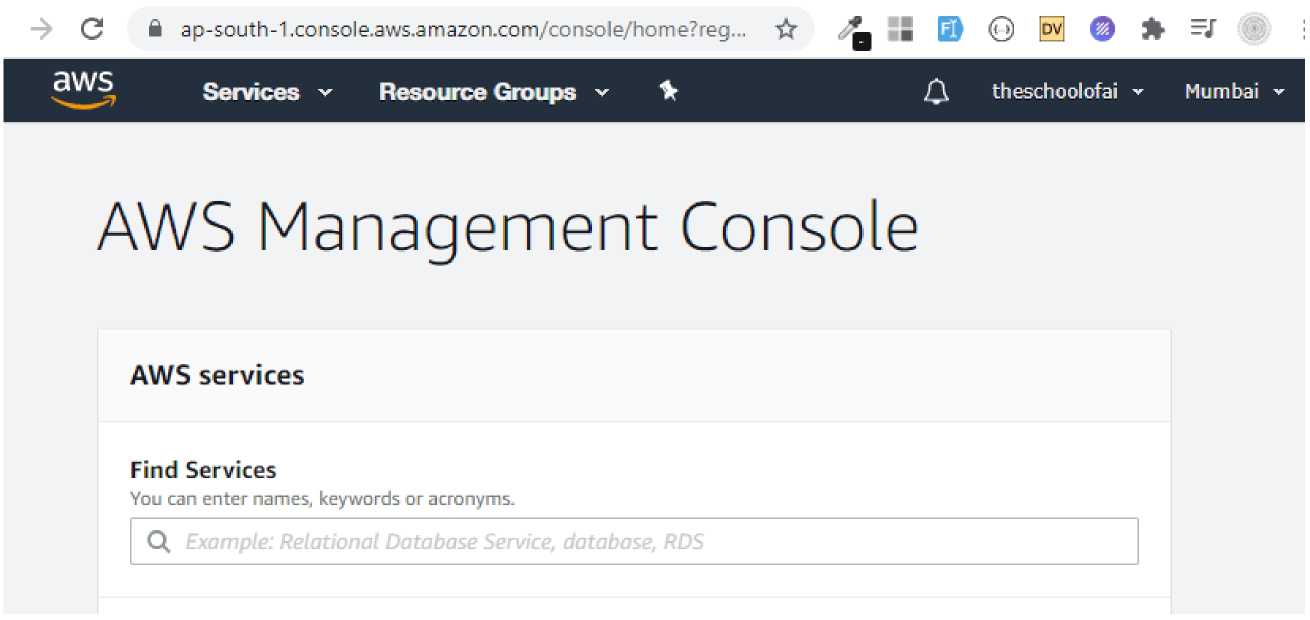
2. check is serverless is installed properly:

   serverless -help

Before we move on to the next topic, let's take another short **quiz (https://s.surveyplanet.com/DuaHaFpHHB)** .

CONFIGURING AWS

1. Sign in to your AWS Console:



AWS Management Console

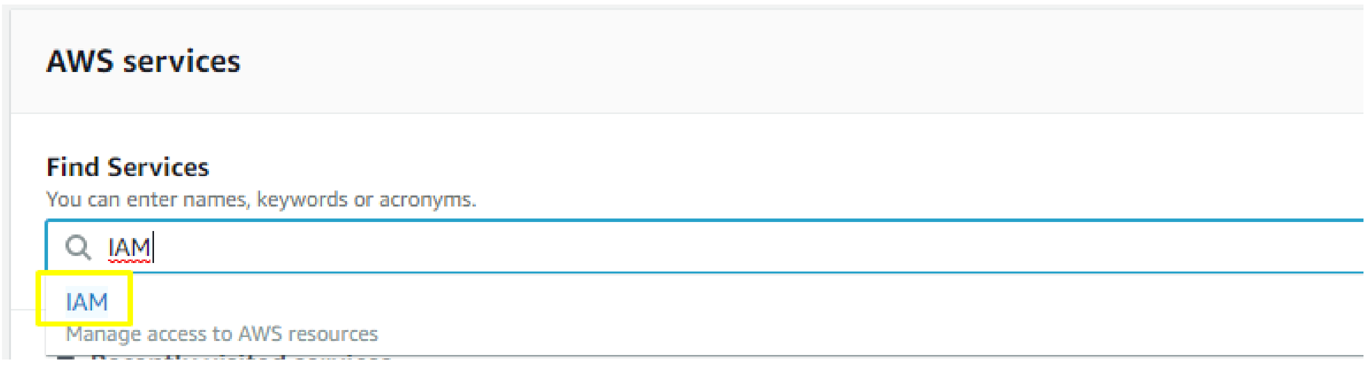
AWS services

Find Services

You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

2. Create a new user for programmatic access. Select IAM (Identity and Access Management):



AWS services

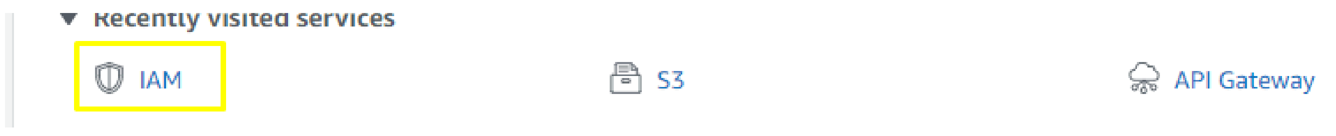
Find Services

You can enter names, keywords or acronyms.

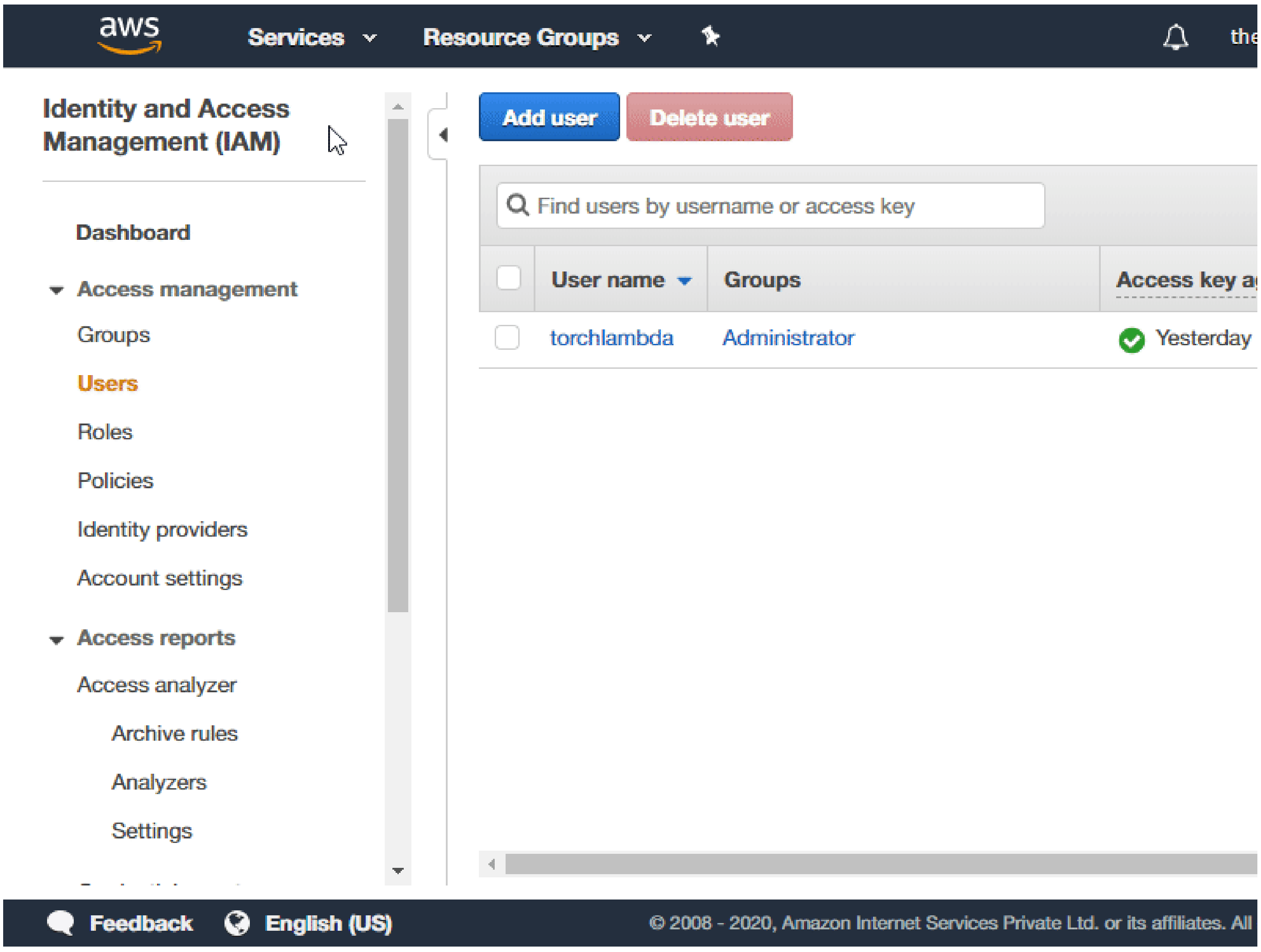
IAM

IAM

Manage access to AWS resources



3. Create a new user:

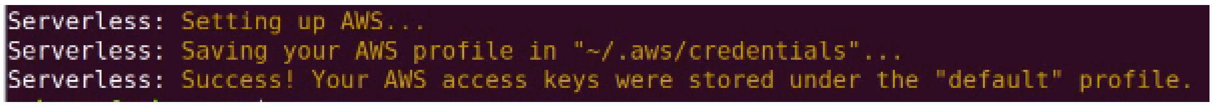


User name	Groups	Access key a
torchlambda	Administrator	✓ Yesterday

- Users, then
- Add User, then
- add a username and access type to "Programmatic Access", this means this user will not be able to login using username and password, but only have programmatic access
- For permissions, Attach existing policies directly.
- Select AdministratorAccess (till Serverless framework comes with something better)
- Select Create User
- **Download .csv file.** This is the only time opportunity to download these credentials.
- You're ready to set up serverless Framework

Setting up SLS with the user credentials

1. Configure Serverless (using your key and secret):
   $ **sls config credentials -- provider aws -- key AKIBWDBSFP1ACHJPAAGQ --secret PZWjxph8vFUkvLinL0frtBk1NijOKjl18DjFMEqm**



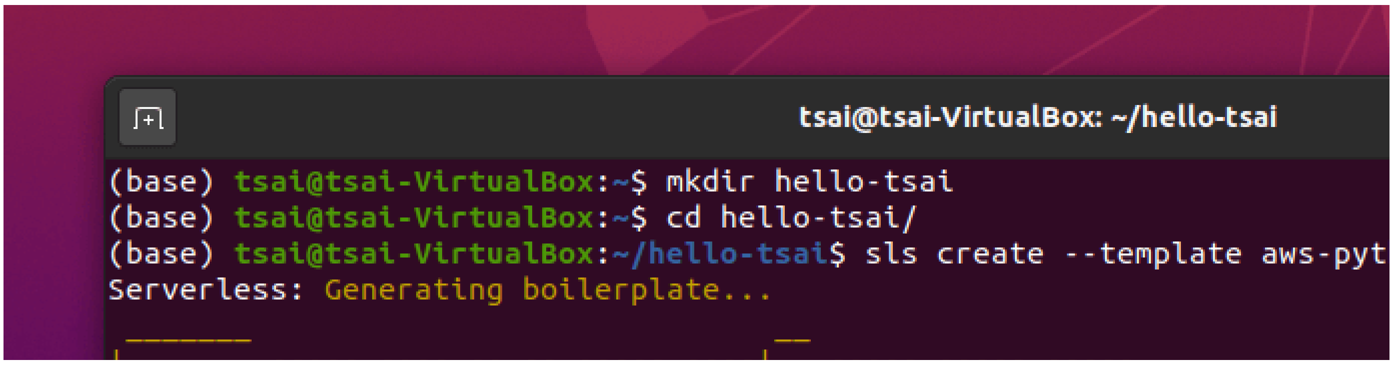
```
Serverless: Setting up AWS...
Serverless: Saving your AWS profile in "~/.aws/credentials"...
Serverless: Success! Your AWS access keys were stored under the "default" profile.
```

2. done!

HELLO TSAI FROM LAMBDA!

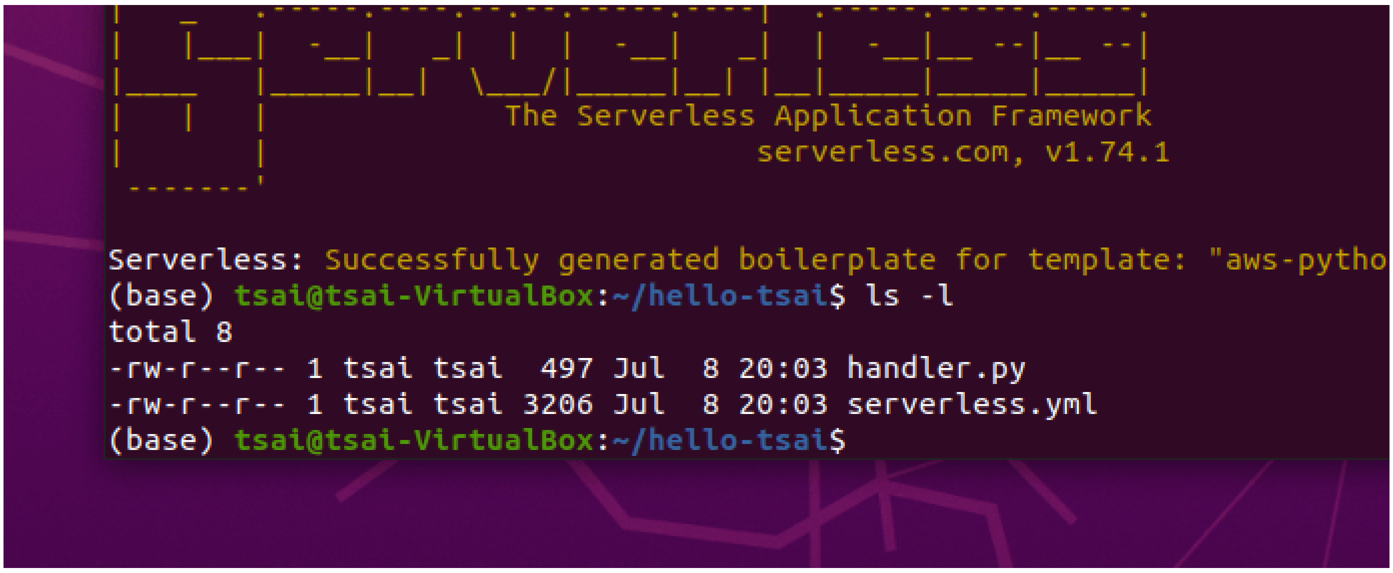
1. Make a new directory for your new project and move into it
2. Execute this command:

   $ sls create --template aws-python3 --name hello-tsai



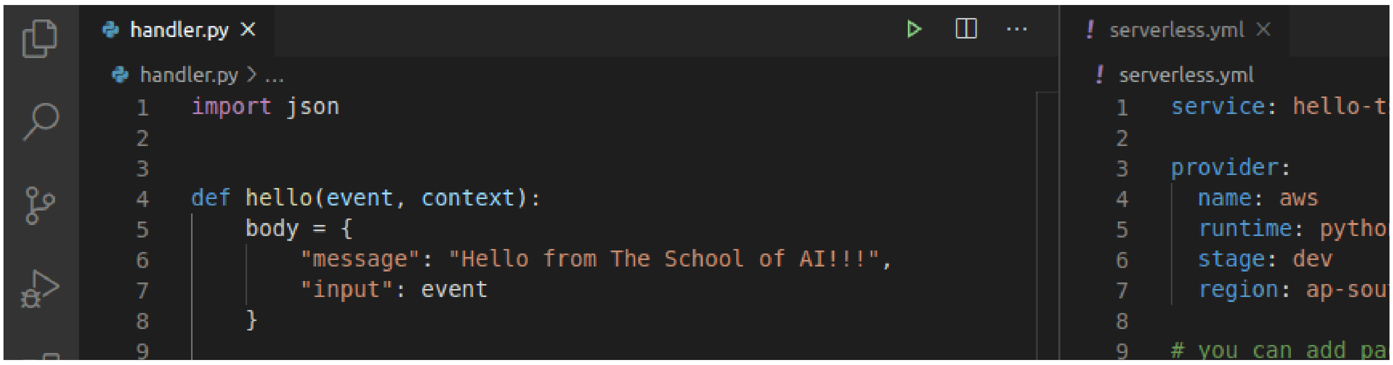
tsai@tsai-VirtualBox: ~/hello-tsai

```
(base) tsai@tsai-VirtualBox:~$ mkdir hello-tsai
(base) tsai@tsai-VirtualBox:~$ cd hello-tsai/
(base) tsai@tsai-VirtualBox:~/hello-tsai$ sls create --template aws-pyt
Serverless: Generating boilerplate...
```



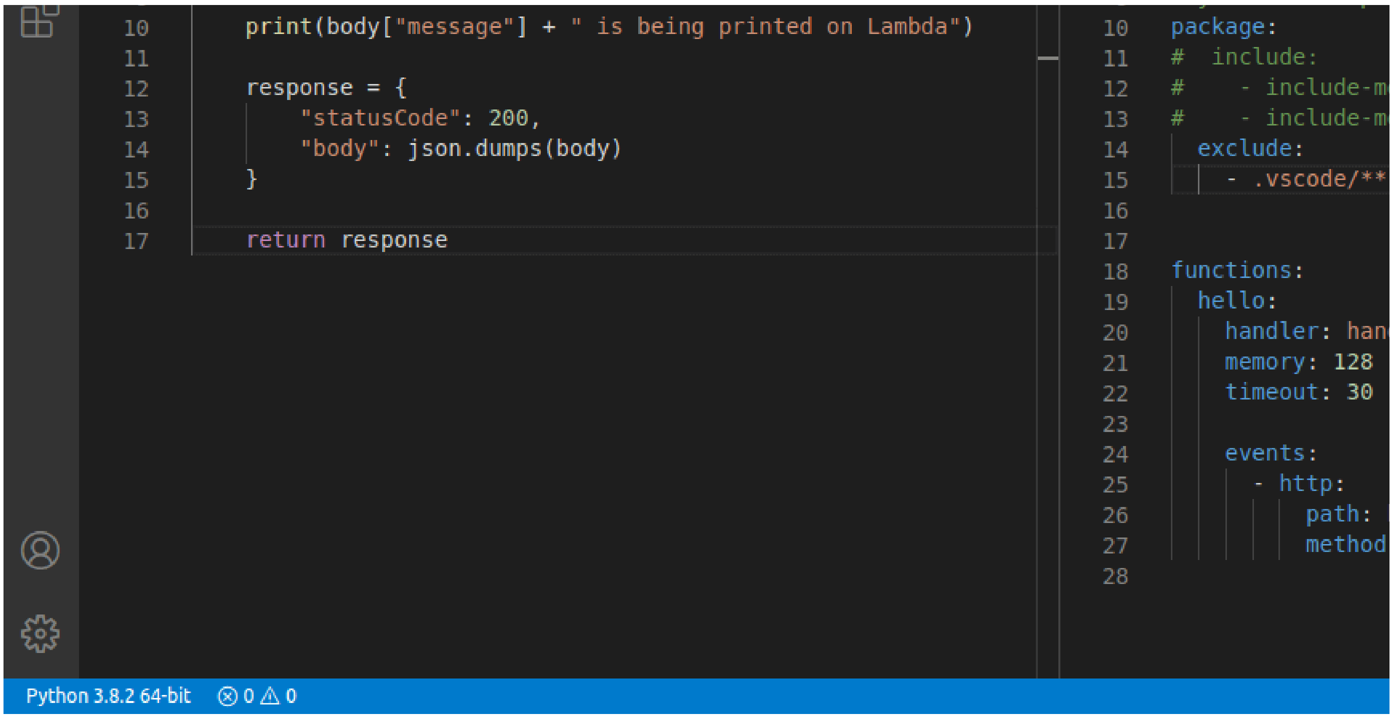
```
|                    The Serverless Application Framework
|                                  serverless.com, v1.74.1

Serverless: Successfully generated boilerplate for template: "aws-pytho
(base) tsai@tsai-VirtualBox:~/hello-tsai$ ls -l
total 8
-rw-r--r-- 1 tsai tsai  497 Jul  8 20:03 handler.py
-rw-r--r-- 1 tsai tsai 3206 Jul  8 20:03 serverless.yml
(base) tsai@tsai-VirtualBox:~/hello-tsai$
```

3. Check the contents of the files we just created.
4. Open the files in VS Code and edit them as shown:



```python
import json


def hello(event, context):
    body = {
        "message": "Hello from The School of AI!!!",
        "input": event
    }
```

```yaml
service: hello-t

provider:
  name: aws
  runtime: pytho
  stage: dev
  region: ap-sou

# you can add pa
```

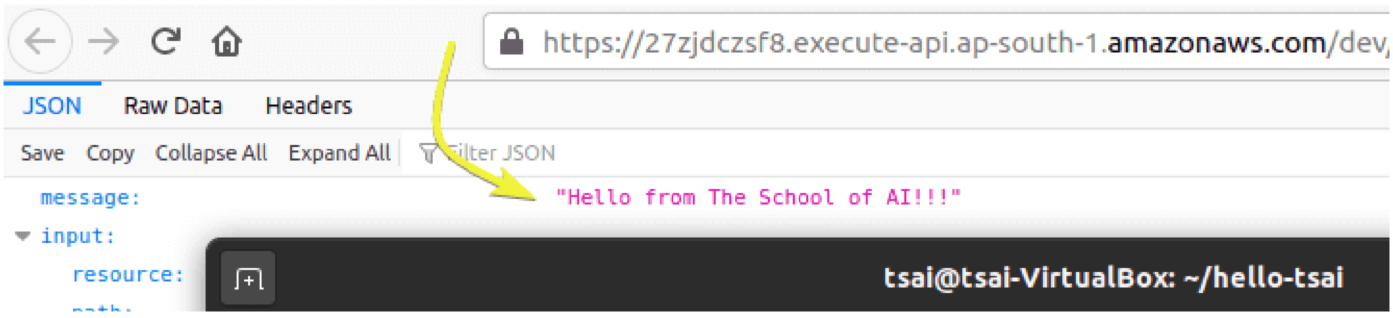


```
    print(body["message"] + " is being printed on Lambda")

    response = {
        "statusCode": 200,
        "body": json.dumps(body)
    }

    return response
```

```
package:
#  include:
#    - include-m
#    - include-m
  exclude:
    - .vscode/**

functions:
  hello:
    handler: han
    memory: 128
    timeout: 30

    events:
      - http:
          path:
          method
```

Python 3.8.2 64-bit ⊗ 0 ⚠ 0

5. You can find regions [here (https://docs.aws.amazon.com/general/latest/gr/rande.html#regional-endpoints)](https://docs.aws.amazon.com/general/latest/gr/rande.html#regional-endpoints) .
6. Just
   $ sls deploy



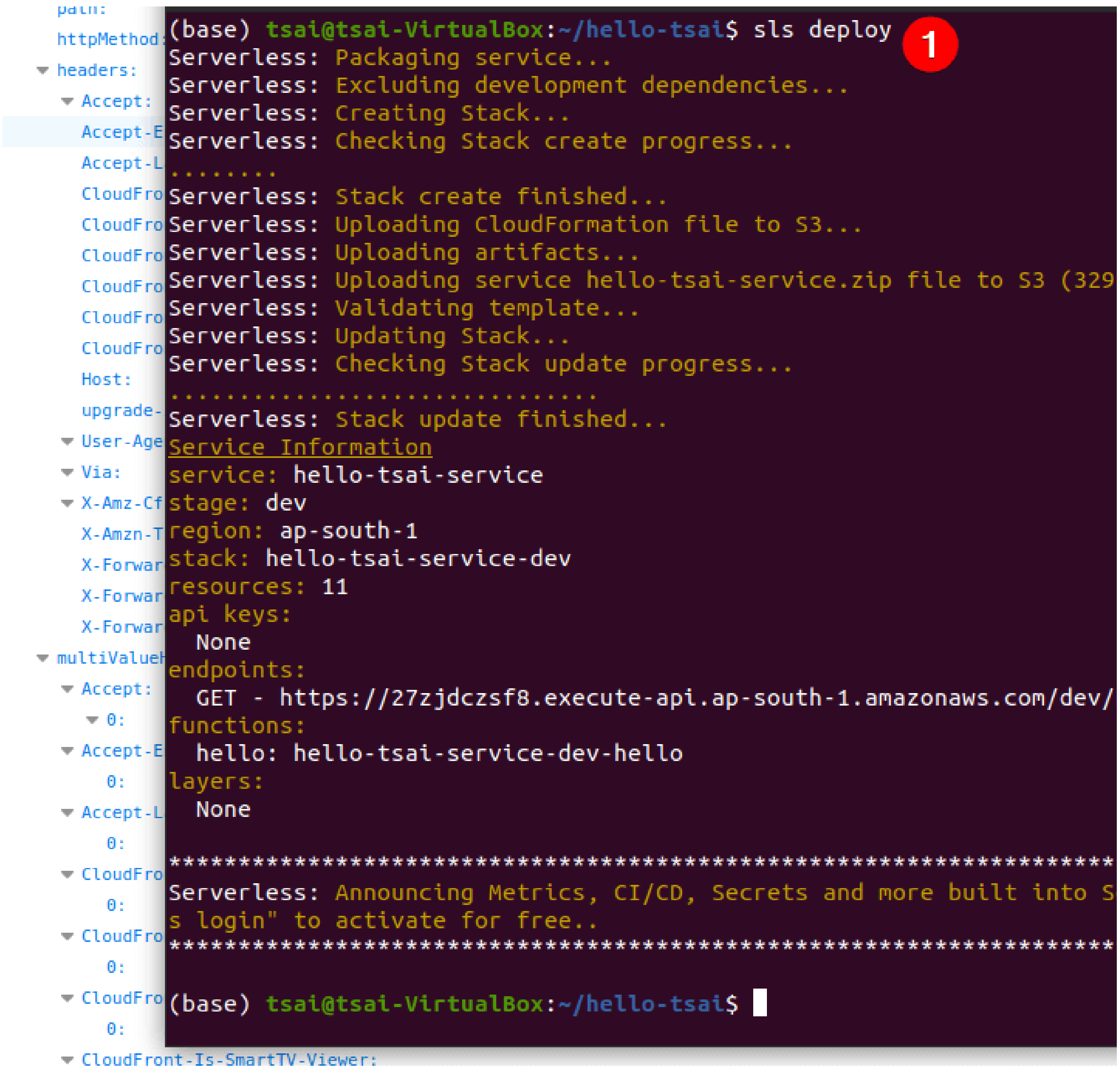
https://27zjdczsf8.execute-api.ap-south-1.amazonaws.com/dev

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
message:    "Hello from The School of AI!!!"
▼ input:
    resource:
```

tsai@tsai-VirtualBox: ~/hello-tsai

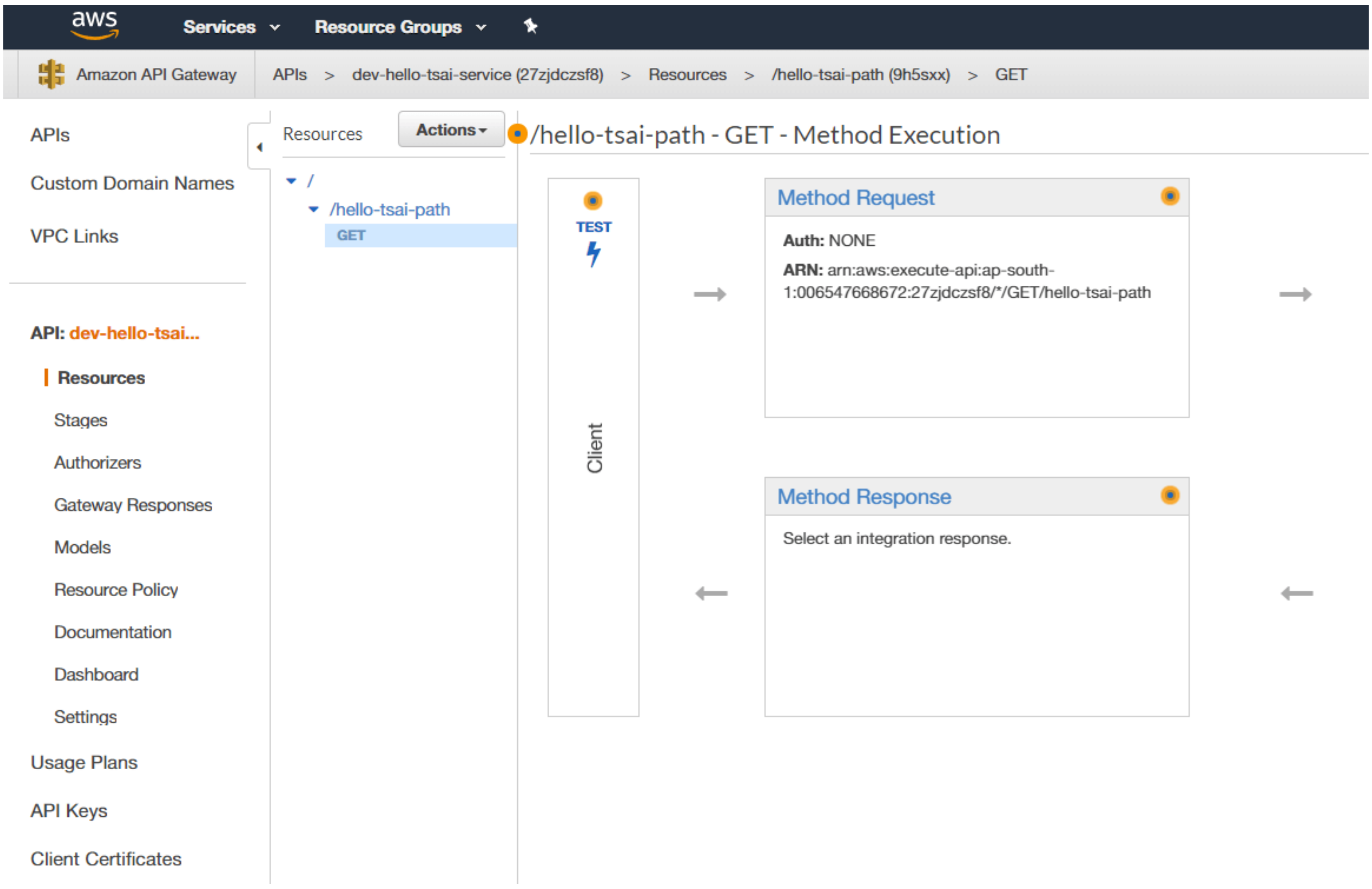


```
(base) tsai@tsai-VirtualBox:~/hello-tsai$ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Creating Stack...
Serverless: Checking Stack create progress...
........
Serverless: Stack create finished...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service hello-tsai-service.zip file to S3 (329
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
...............................
Serverless: Stack update finished...
Service Information
service: hello-tsai-service
stage: dev
region: ap-south-1
stack: hello-tsai-service-dev
resources: 11
api keys:
  None
endpoints:
  GET - https://27zjdczsf8.execute-api.ap-south-1.amazonaws.com/dev/
functions:
  hello: hello-tsai-service-dev-hello
layers:
  None

*********************************************************************
Serverless: Announcing Metrics, CI/CD, Secrets and more built into S
s login" to activate for free..
*********************************************************************

(base) tsai@tsai-VirtualBox:~/hello-tsai$
```

7. Here is my lambda (https://27zjdczsf8.execute-api.ap-south-1.amazonaws.com/dev/hello-tsai-path).

WHAT JUST HAPPENED?

Go to API Gateway in AWS Management Console. Select dev-hello-tsai-service or your differently named service.

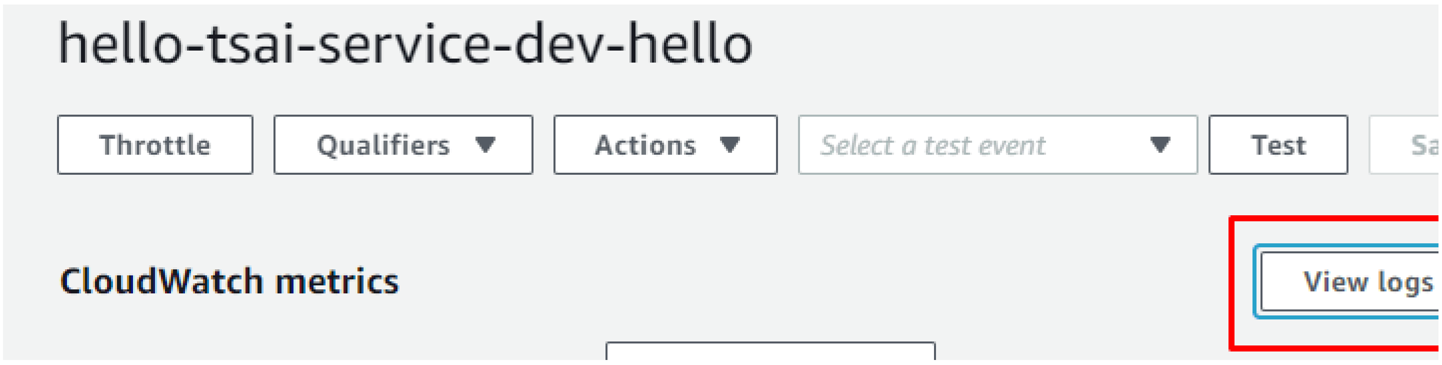


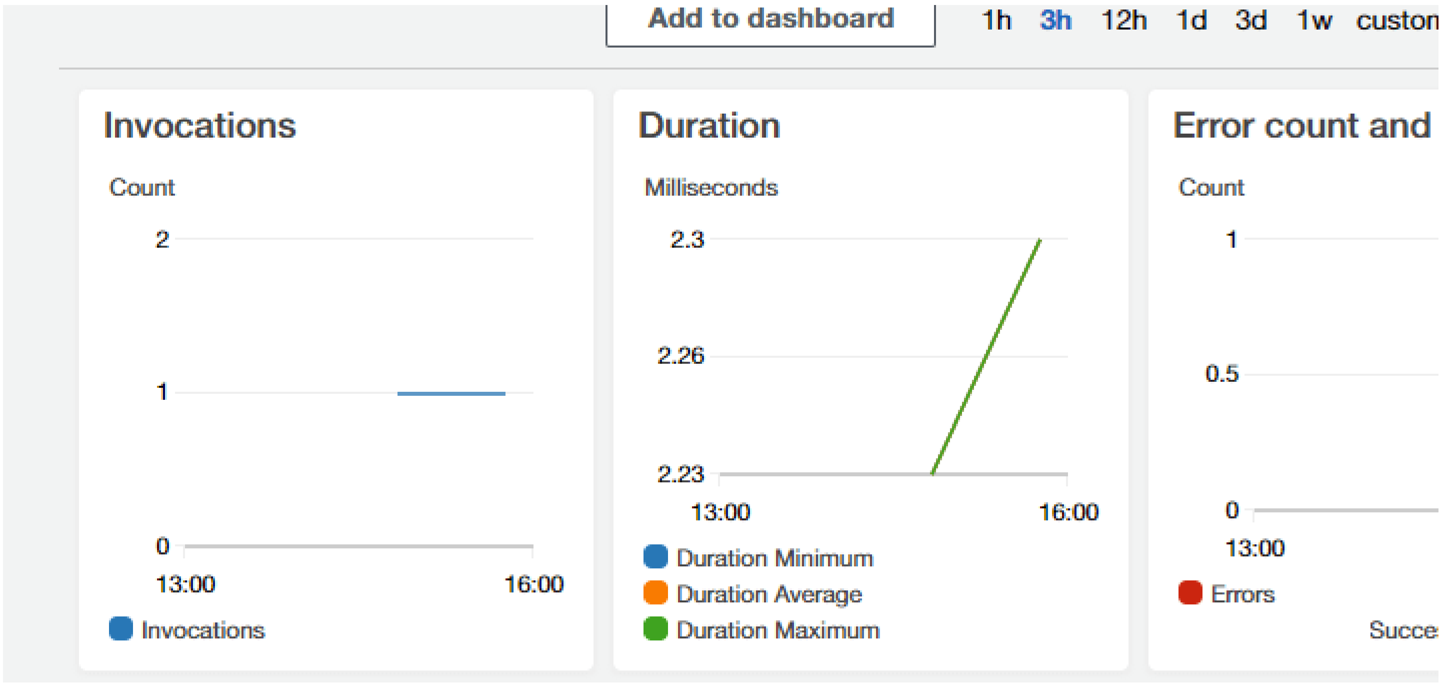
Auth: NONE
ARN: arn:aws:execute-api:ap-south-1:006547668672:27zjdczsf8/*/GET/hello-tsai-path

Select an integration response.

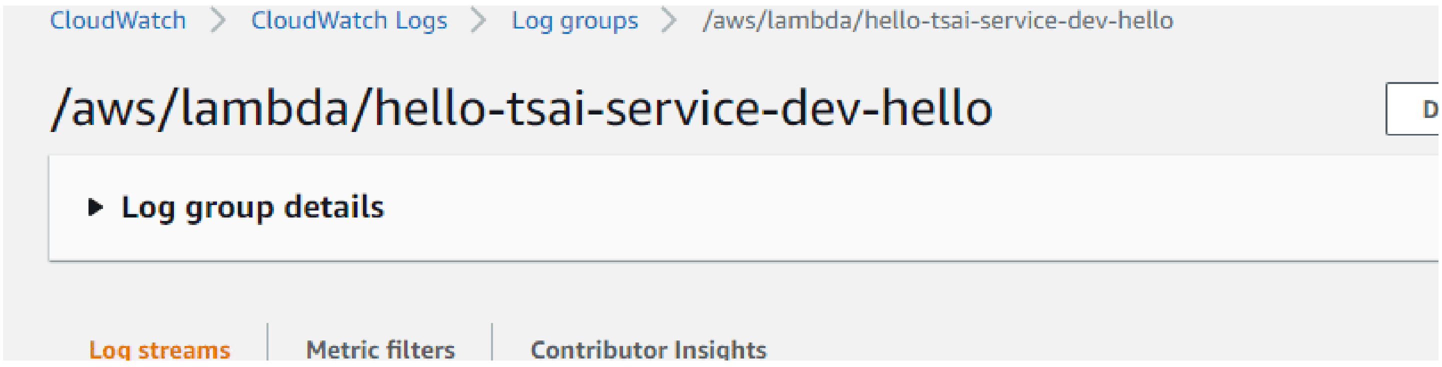
Click on **hello-tsai-service-dev-hello** on the right end and learn more on that page w.r.t. setting you can change/modify.

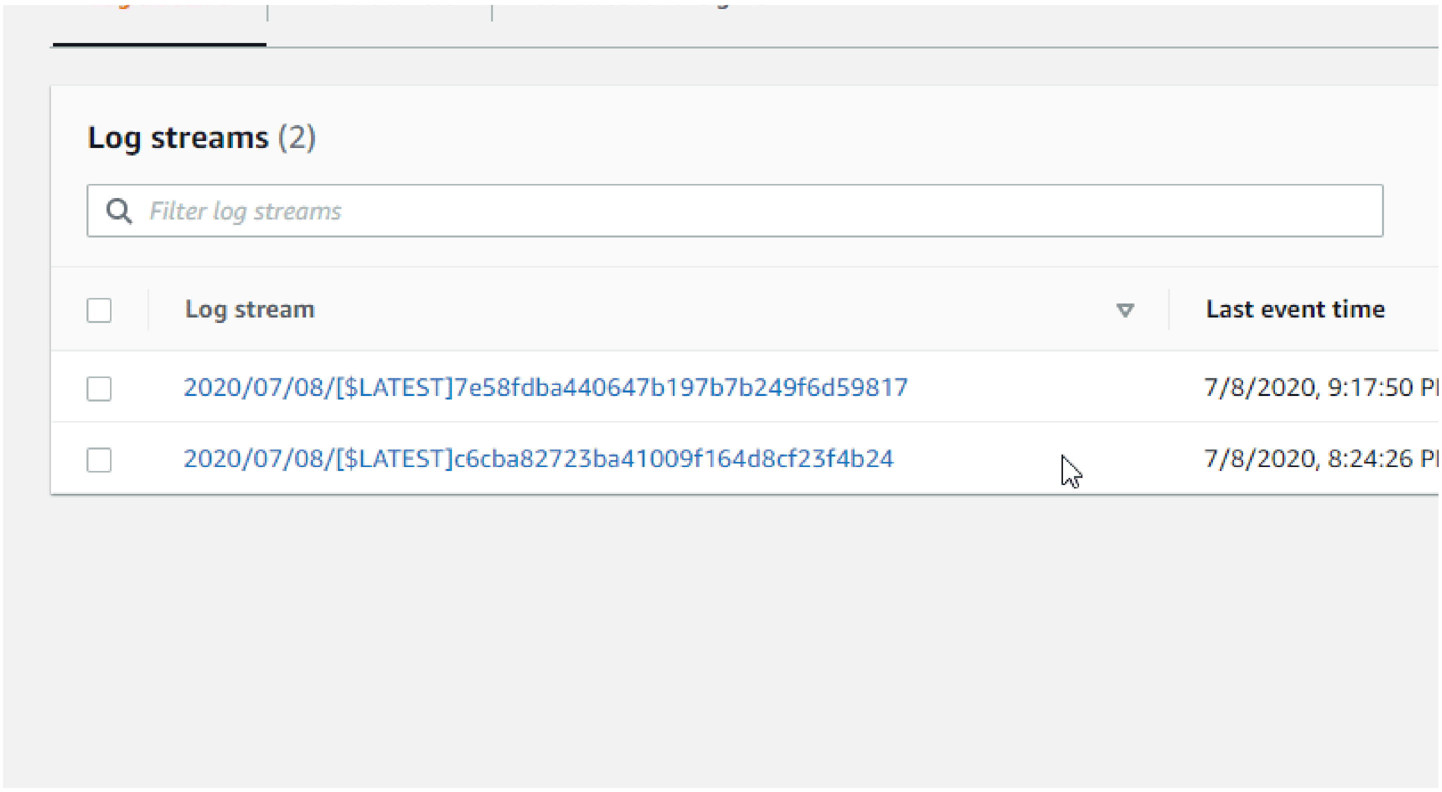
Click on **Monitoring** to access CloudWatch. Here you can see how many times your lambda was accessed and other statistics.





Click on "View logs in CloudWatch"


CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/hello-tsai-service-dev-hello
/aws/lambda/hello-tsai-service-dev-hello
D
▶ Log group details
Log streams
Metric filters
Contributor Insights

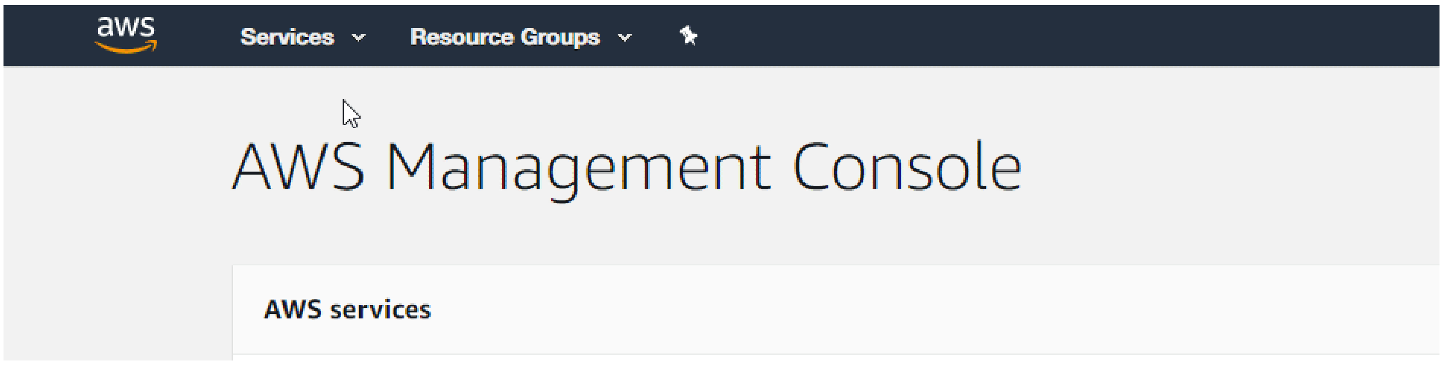


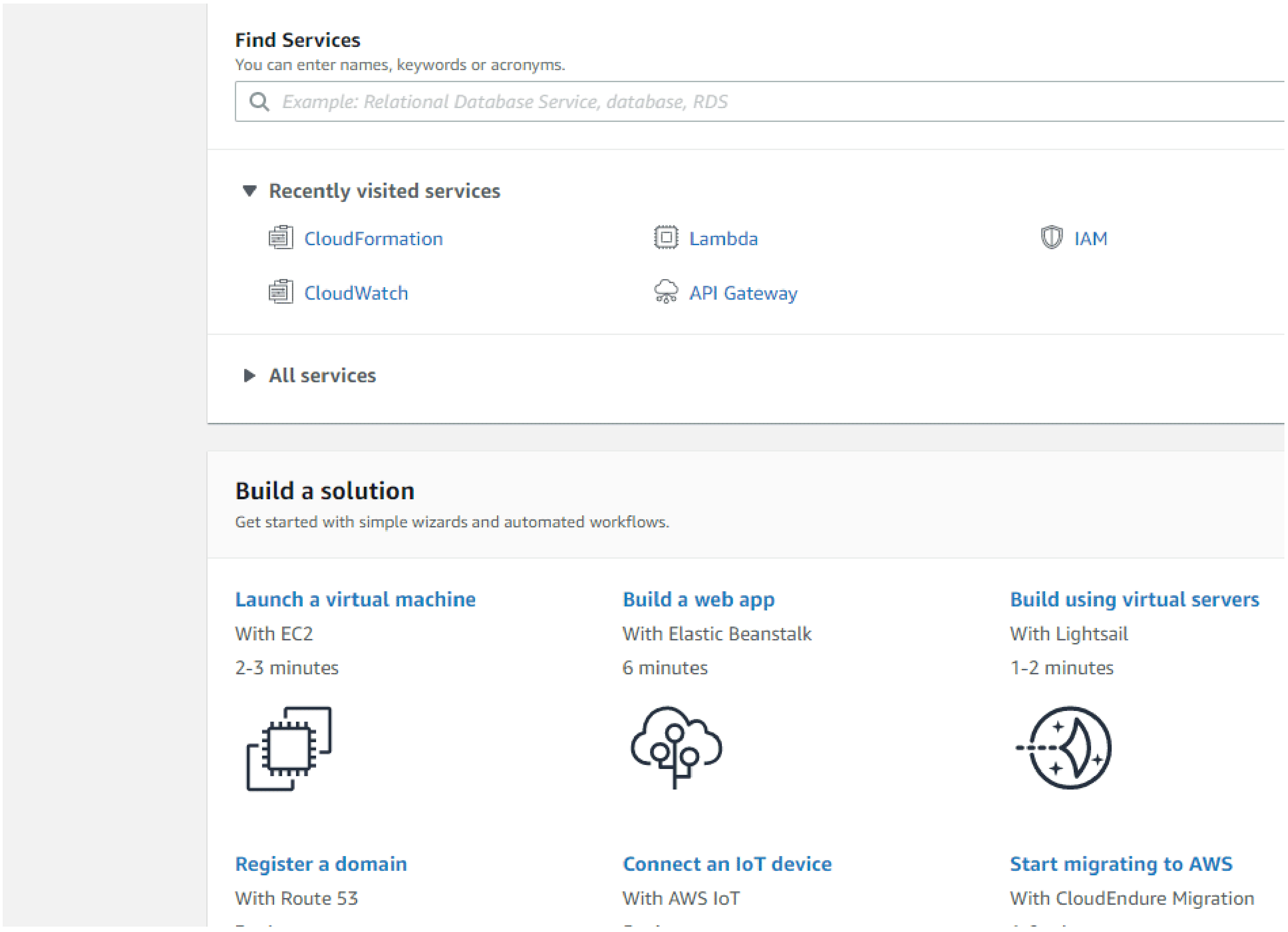
Log streams (2)

Filter log streams

	Log stream	Last event time
☐	2020/07/08/[$LATEST]7e58fdba440647b197b7b249f6d59817	7/8/2020, 9:17:50 PI
☐	2020/07/08/[$LATEST]c6cba82723ba41009f164d8cf23f4b24	7/8/2020, 8:24:26 PI

Let us check out Cloud Formation now. Visit CloudFormation on your AWS console.





Find Services
You can enter names, keywords or acronyms.
Example: Relational Database Service, database, RDS

Recently visited services
CloudFormation
Lambda
IAM
CloudWatch
API Gateway

All services

Build a solution
Get started with simple wizards and automated workflows.

Launch a virtual machine
With EC2
2-3 minutes

Build a web app
With Elastic Beanstalk
6 minutes

Build using virtual servers
With Lightsail
1-2 minutes

Register a domain
With Route 53

Connect an IoT device
With AWS IoT

Start migrating to AWS
With CloudEndure Migration

If we do not use Serverless, you'll have to work with all of these services!

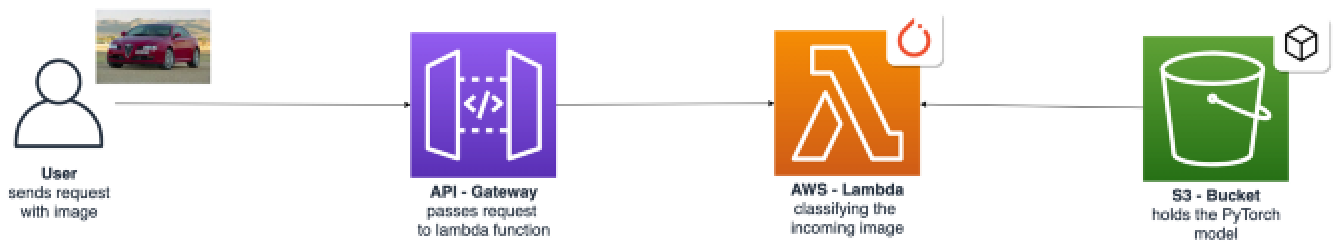
DEPLOYING A DNN MODEL

Deploying a PyTorch model on Lambda is hard, and there are just too many points where one can screw up.

For the first session, we will use a pre-trained Resnet model and then use it to classify our images. We are referencing this amazing **blog (https://towardsdatascience.com/scaling-machine-learning-from-zero-to-hero-d63796442526)** post, so in case you are on Medium, do "clap" for him!

We are going to do these things:

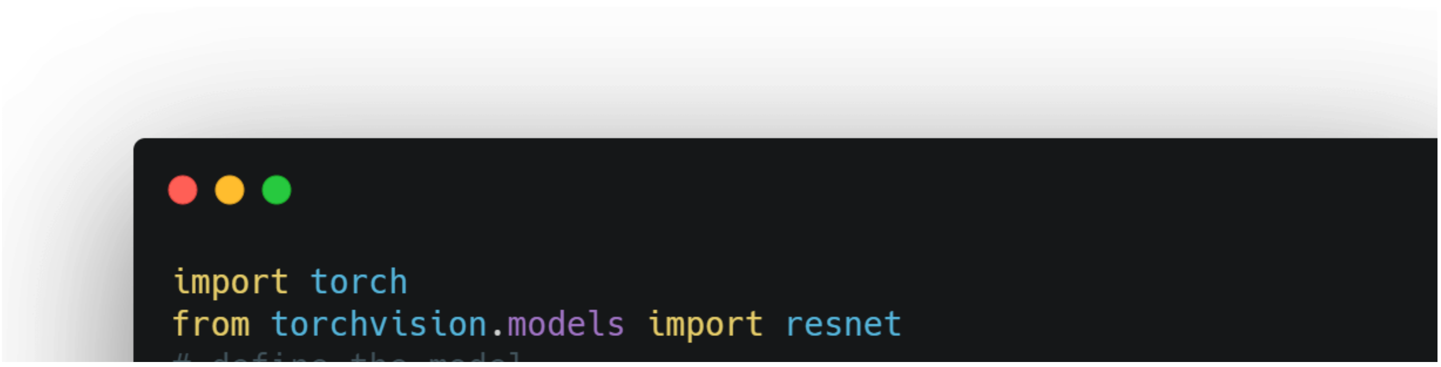
- download a pre-trained ResNet34 model
- create a Python Lambda function with serverless Framework
- create an S3 bucket, which holds the model
- add a Pytorch to the Lambda Environment (the most difficult task)
    - this is going to be our template moving forward for all Lambda Assignments
- write a prediction function to classify an image
- configure the serverless framework to set up the API gateway for inference.
- test our deployment using Insomnia



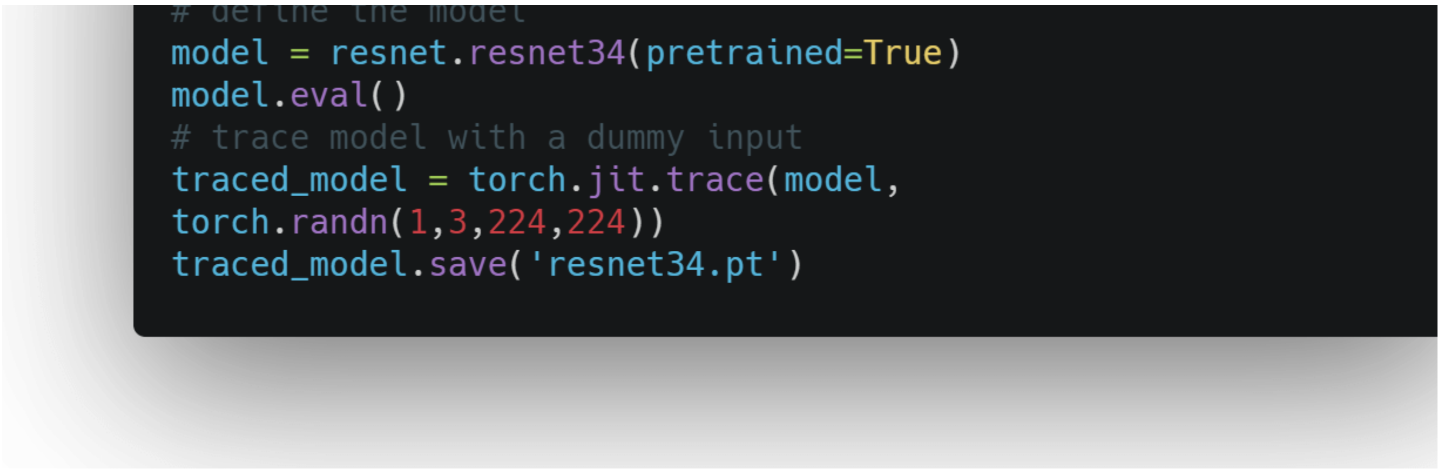
Please make sure that you are in a dedicated Conda environment.

DOWNLOAD A PRETRAINED RESNET34 MODEL

Nothing you don't know here.

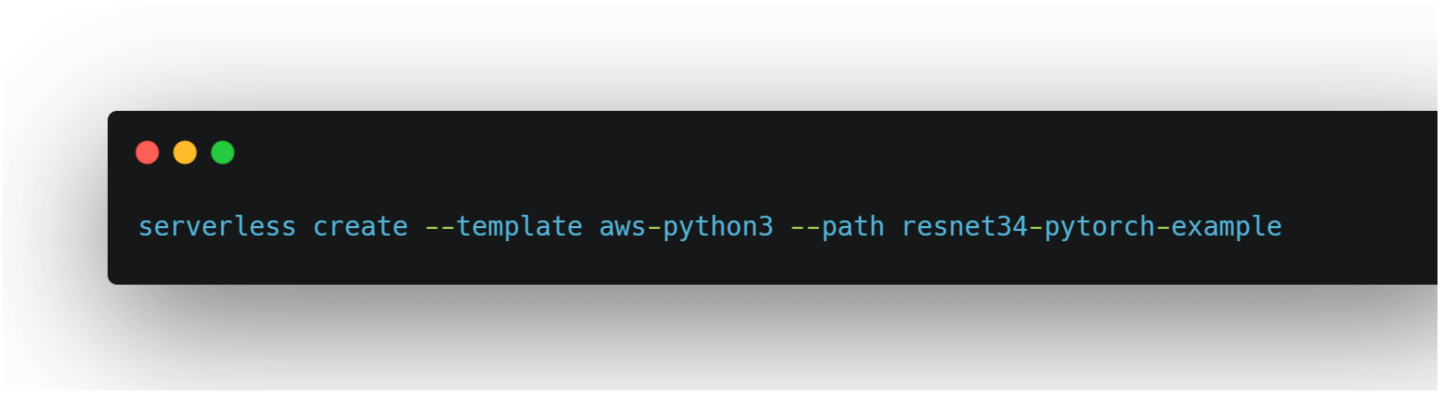


```
import torch
from torchvision.models import resnet
```



```
# define the model
model = resnet.resnet34(pretrained=True)
model.eval()
# trace model with a dummy input
traced_model = torch.jit.trace(model,
torch.randn(1,3,224,224))
traced_model.save('resnet34.pt')
```

CREATE A PYTHON LAMBDA FUNCTION



```
serverless create --template aws-python3 --path resnet34-pytorch-example
```

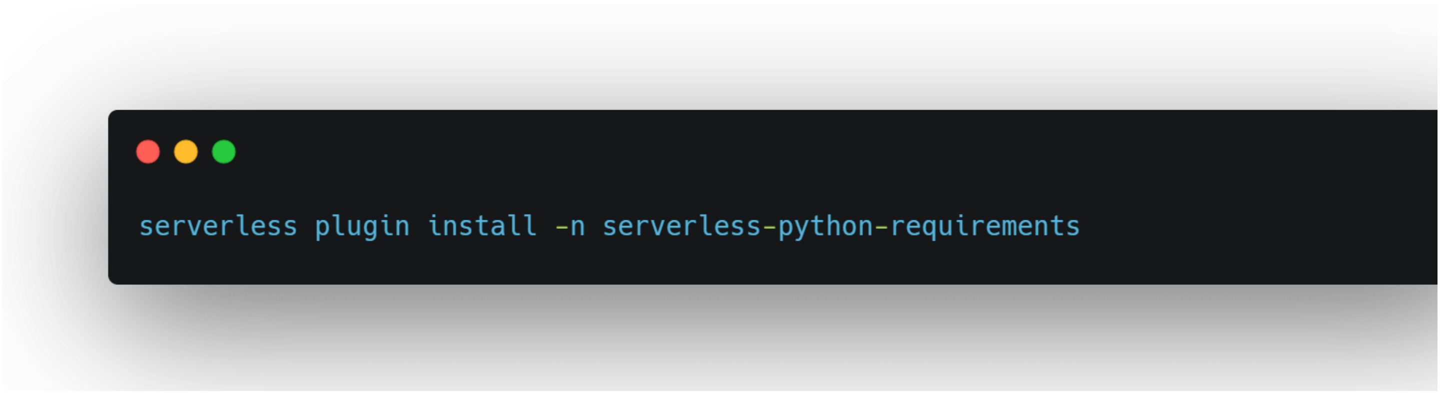
This will generate two main files we'd need to edit, handler.py and 'serverless.yml'

PYTHON REQUIREMENTS PLUGIN

We need to install a Serverless plugin which will automatically bundle dependencies from a requirement.txt file. This is THE main step where we most probably will get stuck (bunding things correctly).

The `serverless-python-requirements` plugin allows you to even bundle non-pure-Python modules. More on that **here** **(https://github.com/UnitedIncome/serverless-python-**

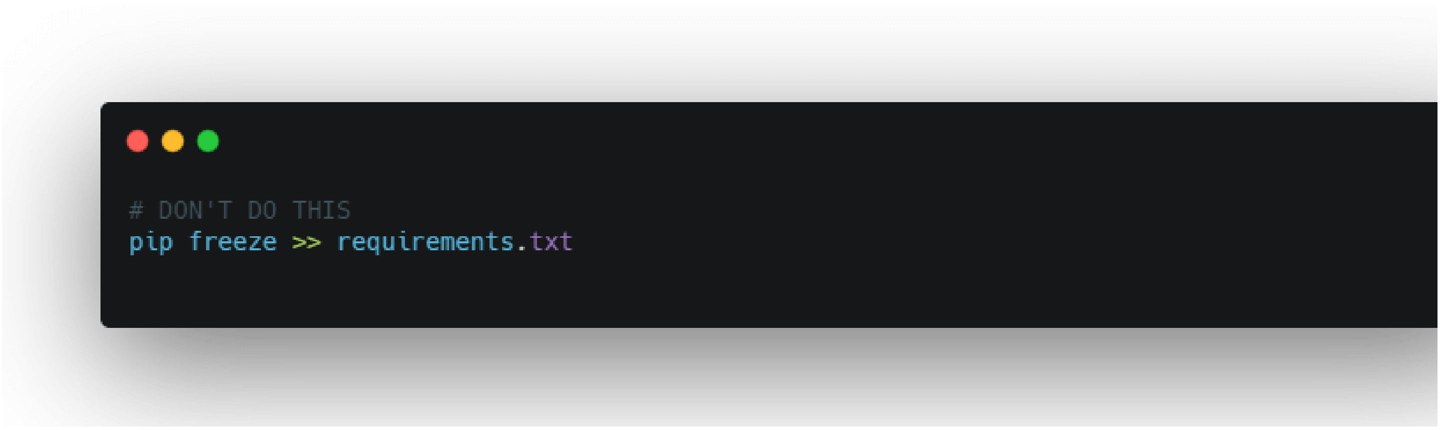
requirements#readme) .



```
serverless plugin install -n serverless-python-requirements
```

ADDING REQUIREMENTS

We'll manually create a requirements.txt file on the root level, with all required Python packages. Normally this file is created using this process, **but don't do this**:

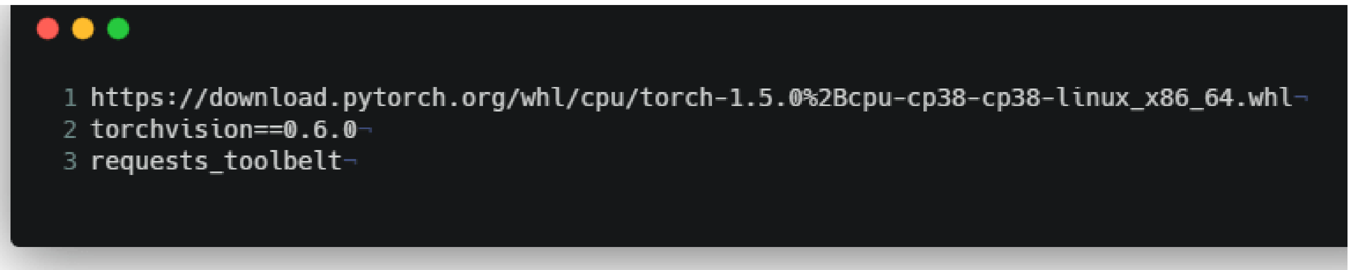


```
# DON'T DO THIS
pip freeze >> requirements.txt
```

We will need to be cautious on how we create this requirement file since our deployment Package on AWS Lambda cannot be greater than 250MB (Pytorch itself can be 470MB or more!).

Instead, we will add a link to python wheel file (.whl) for Pytorch and Lambda will directly install it for us! For a list of all PyTorch and torchvision packages consider this list (https://download.pytorch.org/whl/torch_stable.html) .

The requirements.txt should look like this.



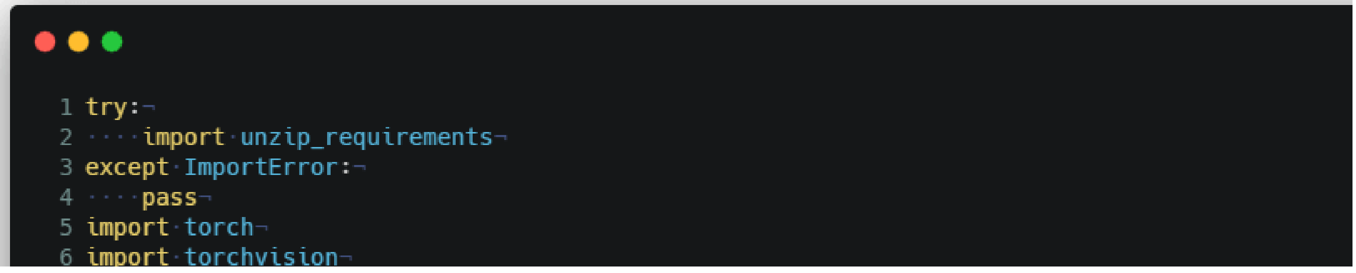
```
https://download.pytorch.org/whl/cpu/torch-1.5.0%2Bcpu-cp38-cp38-linux_x86_64.whl
torchvision==0.6.0
requests_toolbelt
```

To make the dependencies even smaller we will employ three techniques available in the serverless-python-requirements plugin:

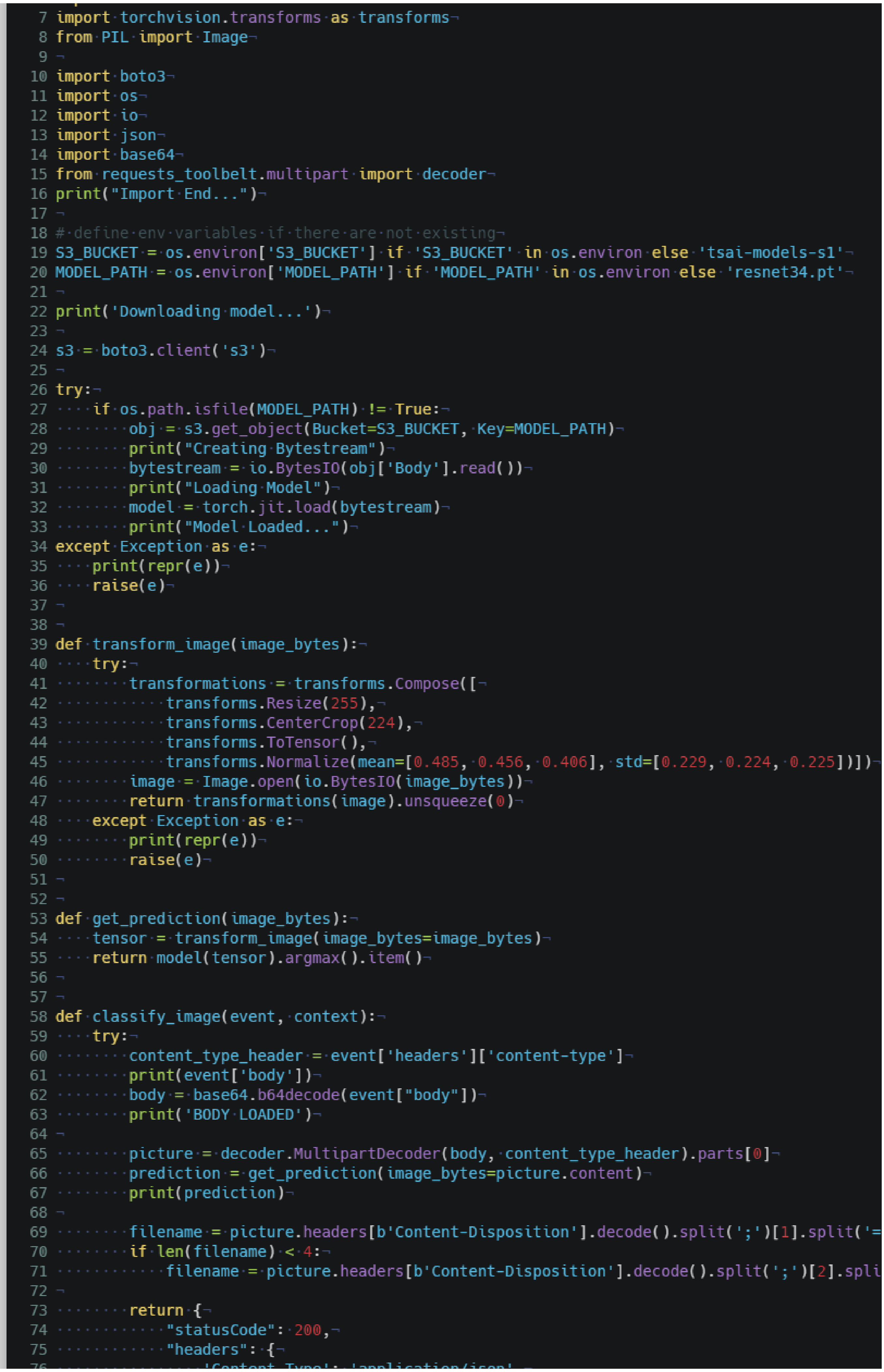
- **zip** - Compresses the dependencies in the requirements.txt in an additional **.requirements.zip** file and adds **unzip_requirements.py** in the final bundle.
- **slim** - Removes unneeded files and directories such as *.so, *.pyc, dist-info, etc.
- **noDeploy** - Omits certain packages from deployment. We will use the standard list that excludes packages those already built into Lambda, as well as Tensorboard.

You can see the implementation of it in the section where we are “configuring our serverless.yaml " file.

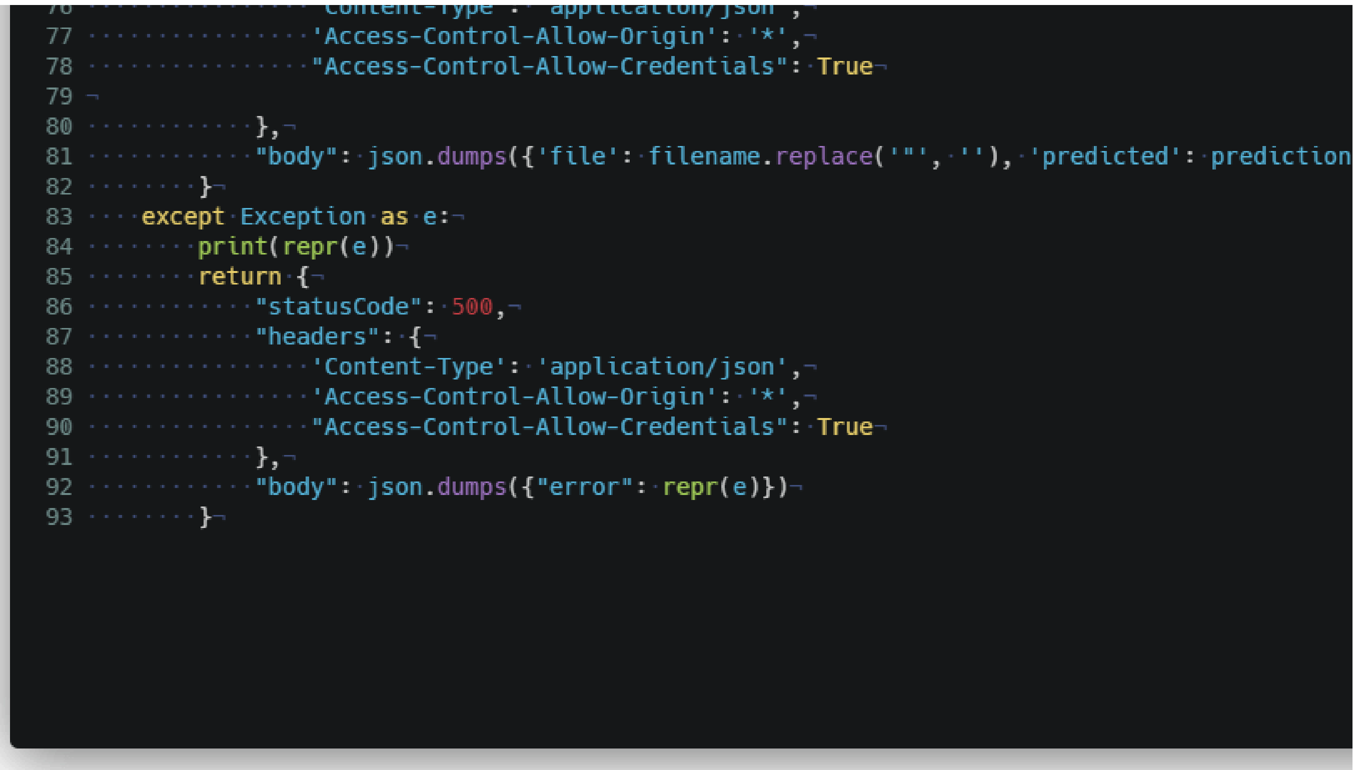
HANDLER.PY AND OUR MAIN CODE



```
try:
    import unzip_requirements
except ImportError:
    pass
import torch
import torchvision
```

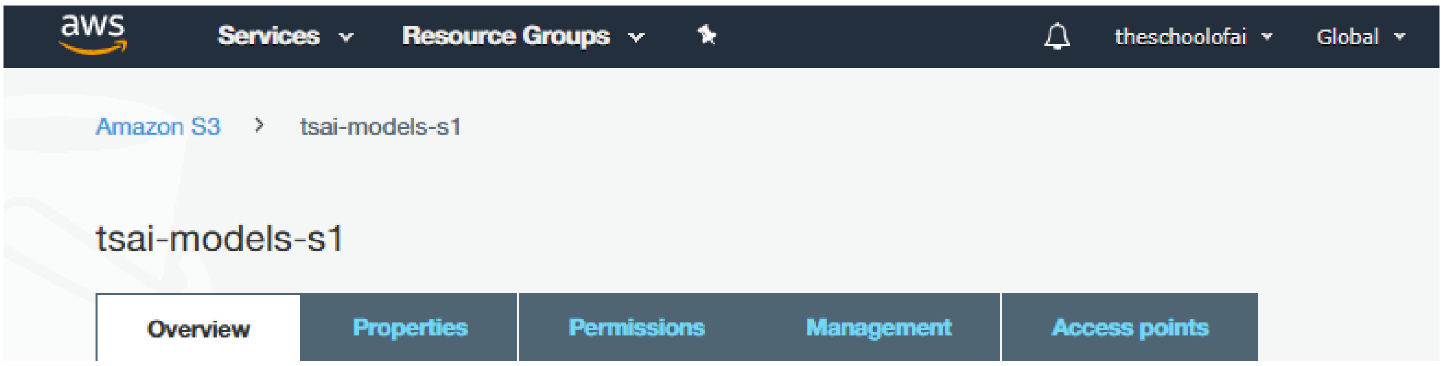


```python
import torchvision.transforms as transforms
from PIL import Image

import boto3
import os
import io
import json
import base64
from requests_toolbelt.multipart import decoder
print("Import End...")

# define env variables if there are not existing
S3_BUCKET = os.environ['S3_BUCKET'] if 'S3_BUCKET' in os.environ else 'tsai-models-s1'
MODEL_PATH = os.environ['MODEL_PATH'] if 'MODEL_PATH' in os.environ else 'resnet34.pt'

print('Downloading model...')

s3 = boto3.client('s3')

try:
    if os.path.isfile(MODEL_PATH) != True:
        obj = s3.get_object(Bucket=S3_BUCKET, Key=MODEL_PATH)
        print("Creating Bytestream")
        bytestream = io.BytesIO(obj['Body'].read())
        print("Loading Model")
        model = torch.jit.load(bytestream)
        print("Model Loaded...")
except Exception as e:
    print(repr(e))
    raise(e)


def transform_image(image_bytes):
    try:
        transformations = transforms.Compose([
            transforms.Resize(255),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
        image = Image.open(io.BytesIO(image_bytes))
        return transformations(image).unsqueeze(0)
    except Exception as e:
        print(repr(e))
        raise(e)


def get_prediction(image_bytes):
    tensor = transform_image(image_bytes=image_bytes)
    return model(tensor).argmax().item()


def classify_image(event, context):
    try:
        content_type_header = event['headers']['content-type']
        print(len(event['body']))
        body = base64.b64decode(event["body"])
        print('BODY LOADED')

        picture = decoder.MultipartDecoder(body, content_type_header).parts[0]
        prediction = get_prediction(image_bytes=picture.content)
        print(prediction)

        filename = picture.headers[b'Content-Disposition'].decode().split(';')[1].split('=
        if len(filename) < 4:
            filename = picture.headers[b'Content-Disposition'].decode().split(';')[2].spli

        return {
            "statusCode": 200,
            "headers": {
                'Content-Type': 'application/json',
```

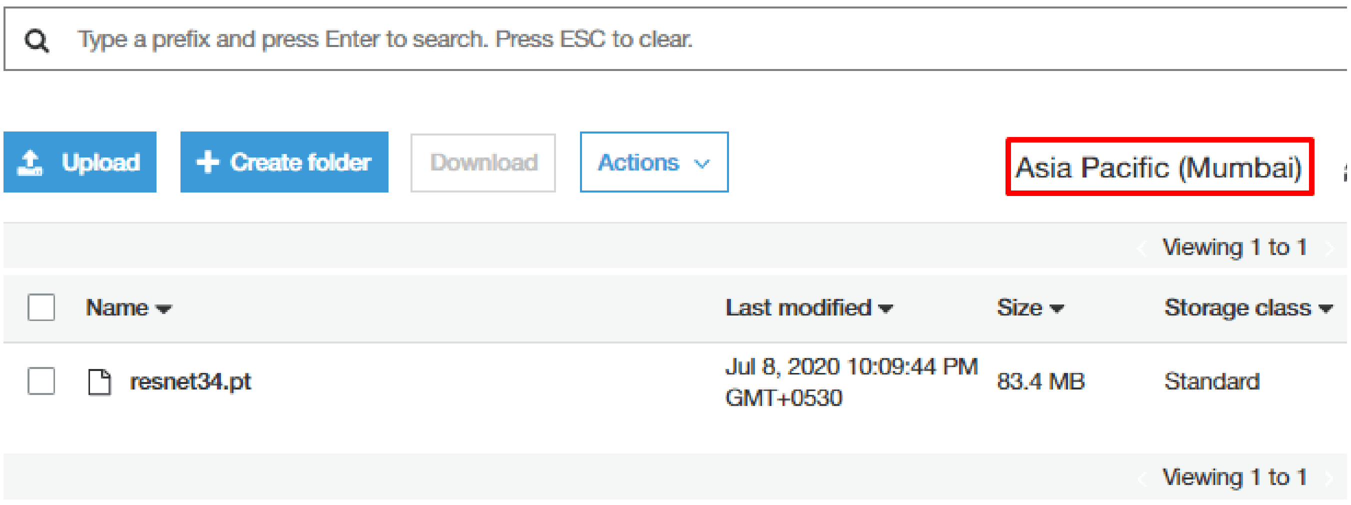


```
                'Access-Control-Allow-Origin': '*',
                "Access-Control-Allow-Credentials": True

            },
            "body": json.dumps({'file': filename.replace('"', ''), 'predicted': prediction
        }
    except Exception as e:
        print(repr(e))
        return {
            "statusCode": 500,
            "headers": {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                "Access-Control-Allow-Credentials": True
            },
            "body": json.dumps({"error": repr(e)})
        }
```

CREATE S3 BUCKET AND MOVE YOUR MODEL THERE

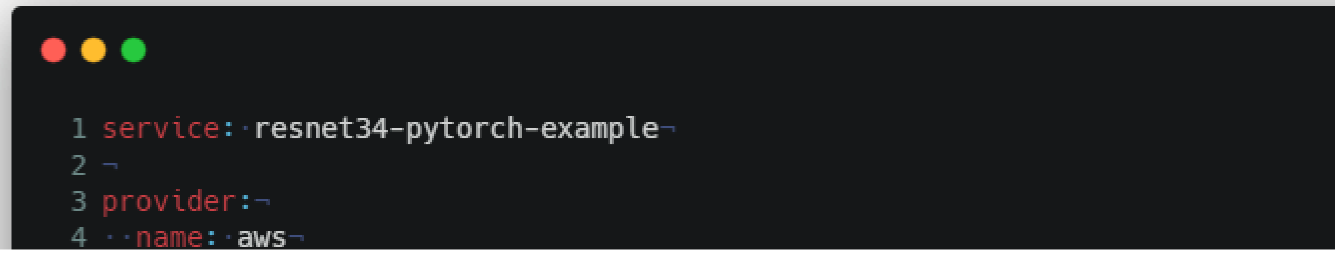


tsai-models-s1

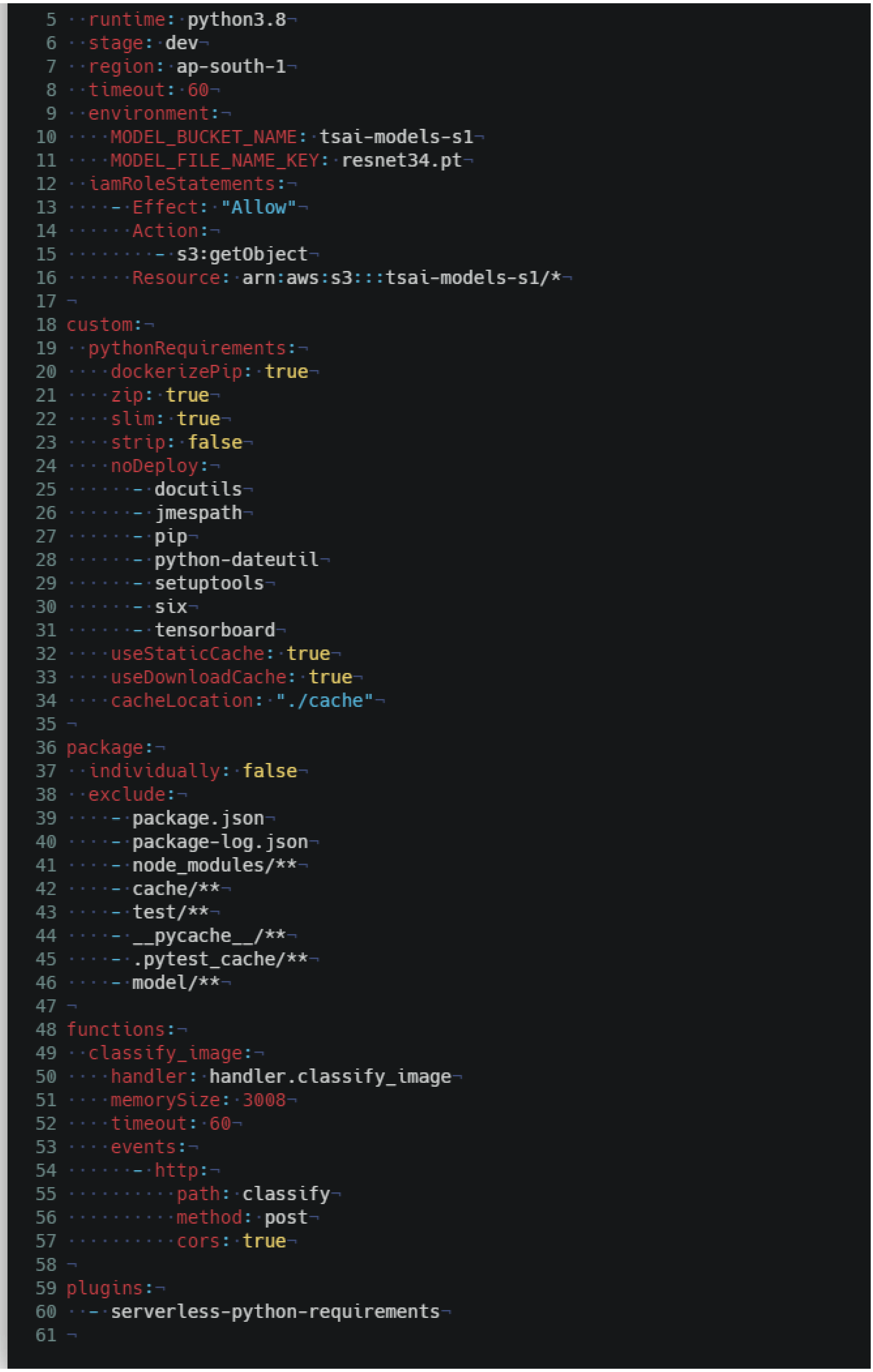


	Name	Last modified	Size	Storage class
	resnet34.pt	Jul 8, 2020 10:09:44 PM GMT+0530	83.4 MB	Standard

CONFIGURING THE SERVERLESS.YML



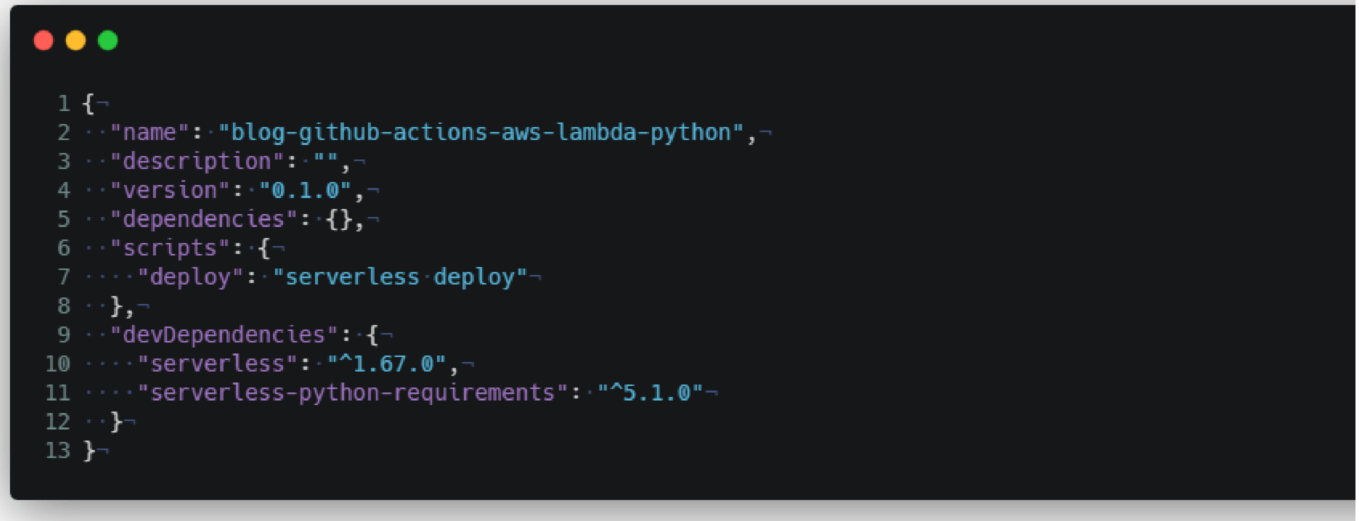
```
service: resnet34-pytorch-example

provider:
  name: aws
```



```yaml
  runtime: python3.8
  stage: dev
  region: ap-south-1
  timeout: 60
  environment:
    MODEL_BUCKET_NAME: tsai-models-s1
    MODEL_FILE_NAME_KEY: resnet34.pt
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - s3:getObject
      Resource: arn:aws:s3:::tsai-models-s1/*

custom:
  pythonRequirements:
    dockerizePip: true
    zip: true
    slim: true
    strip: false
    noDeploy:
      - docutils
      - jmespath
      - pip
      - python-dateutil
      - setuptools
      - six
      - tensorboard
    useStaticCache: true
    useDownloadCache: true
    cacheLocation: "./cache"

package:
  individually: false
  exclude:
    - package.json
    - package-log.json
    - node_modules/**
    - cache/**
    - test/**
    - __pycache__/**
    - .pytest_cache/**
    - model/**

functions:
  classify_image:
    handler: handler.classify_image
    memorySize: 3008
    timeout: 60
    events:
      - http:
          path: classify
          method: post
          cors: true

plugins:
  - serverless-python-requirements

```

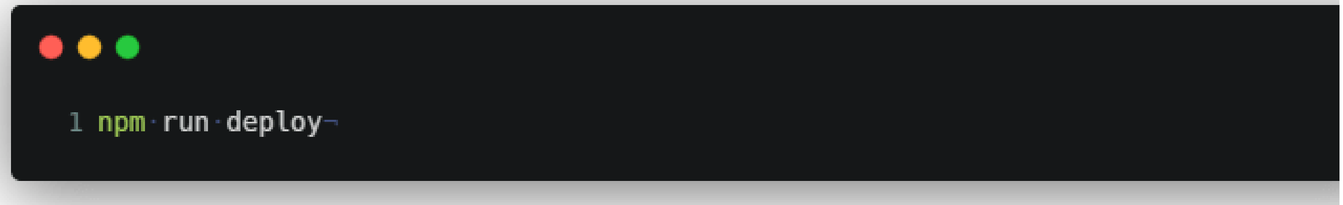
DEPLOYING OUR PACKAGE

Add a deploy script to your package.json file.



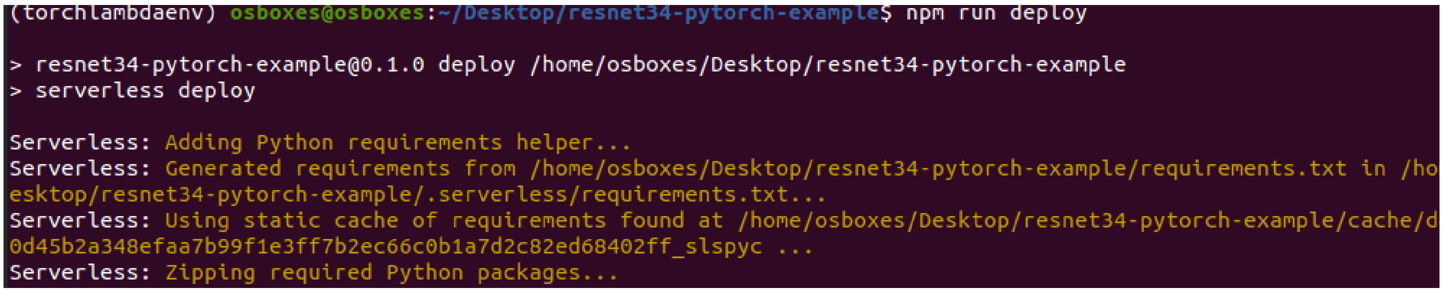
```json
{
  "name": "blog-github-actions-aws-lambda-python",
  "description": "",
  "version": "0.1.0",
  "dependencies": {},
  "scripts": {
    "deploy": "serverless deploy"
  },
  "devDependencies": {
    "serverless": "^1.67.0",
    "serverless-python-requirements": "^5.1.0"
  }
}
```

And then 😎

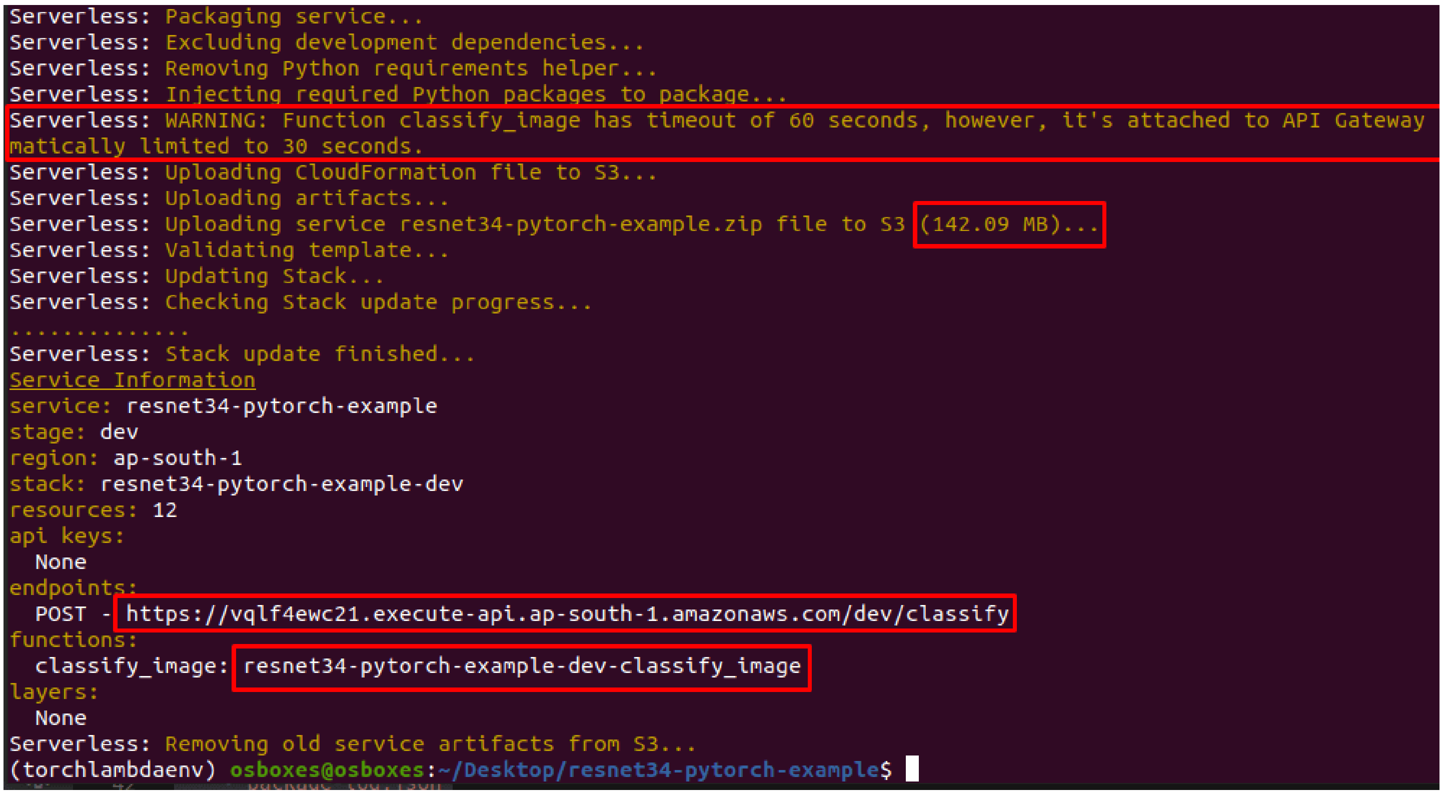


```
npm run deploy
```

You'll get a log like this:



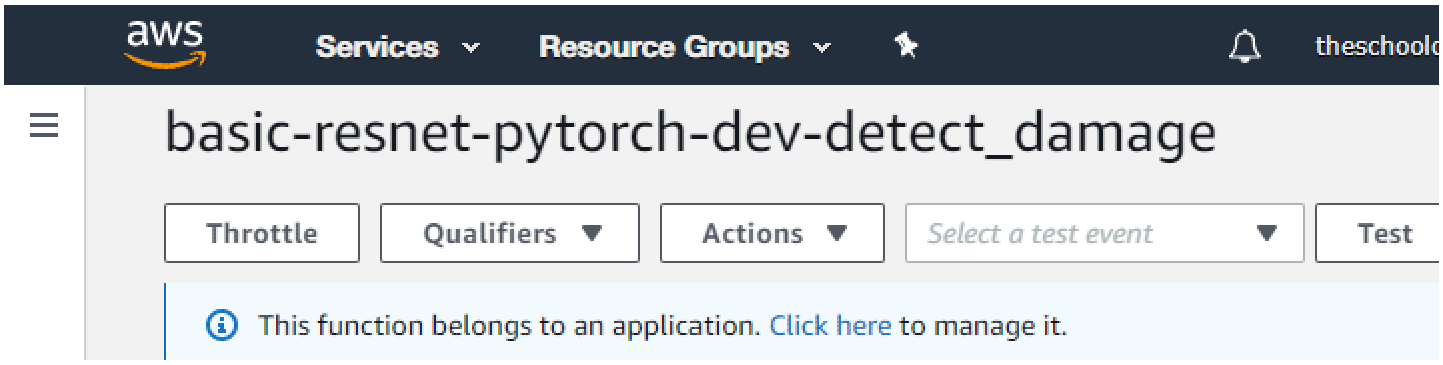
```
(torchlambdaenv) osboxes@osboxes:~/Desktop/resnet34-pytorch-example$ npm run deploy

> resnet34-pytorch-example@0.1.0 deploy /home/osboxes/Desktop/resnet34-pytorch-example
> serverless deploy

Serverless: Adding Python requirements helper...
Serverless: Generated requirements from /home/osboxes/Desktop/resnet34-pytorch-example/requirements.txt in /ho
esktop/resnet34-pytorch-example/.serverless/requirements.txt...
Serverless: Using static cache of requirements found at /home/osboxes/Desktop/resnet34-pytorch-example/cache/d
0d45b2a348efaa7b99f1e3ff7b2ec66c0b1a7d2c82ed68402ff_slspyc ...
Serverless: Zipping required Python packages...
```



```
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Removing Python requirements helper...
Serverless: Injecting required Python packages to package...
Serverless: WARNING: Function classify_image has timeout of 60 seconds, however, it's attached to API Gateway
matically limited to 30 seconds.
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service resnet34-pytorch-example.zip file to S3 (142.09 MB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
..............
Serverless: Stack update finished...
Service Information
service: resnet34-pytorch-example
stage: dev
region: ap-south-1
stack: resnet34-pytorch-example-dev
resources: 12
api keys:
  None
endpoints:
  POST - https://vqlf4ewc21.execute-api.ap-south-1.amazonaws.com/dev/classify
functions:
  classify_image: resnet34-pytorch-example-dev-classify_image
layers:
  None
Serverless: Removing old service artifacts from S3...
(torchlambdaenv) osboxes@osboxes:~/Desktop/resnet34-pytorch-example$
```

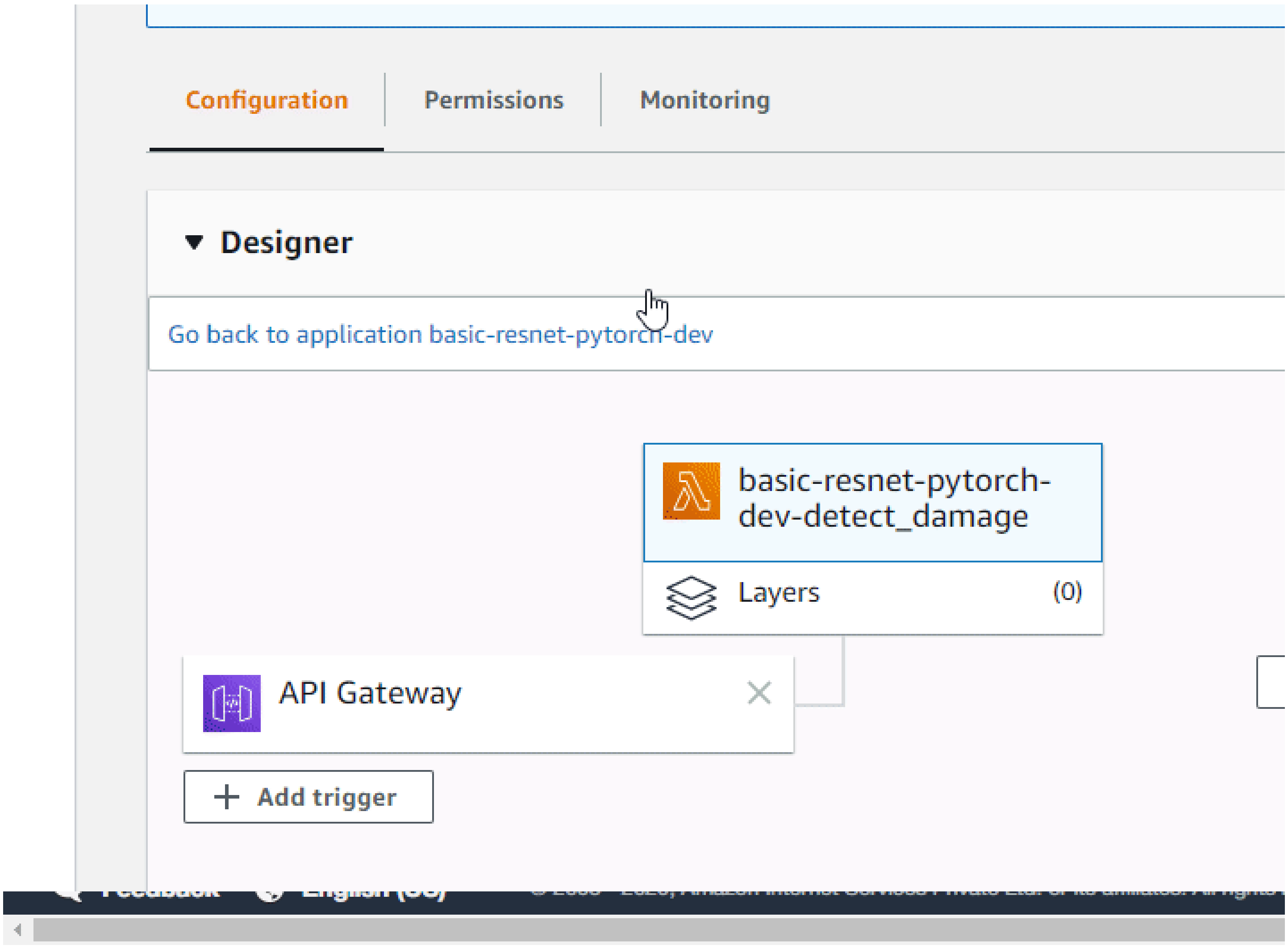
If we were to check our deployment right now, it won't work as we are sending base64 decoded data using multipart/form-data and AWS API gateway isn't told about that.

MULTIPART/FORM-DATA



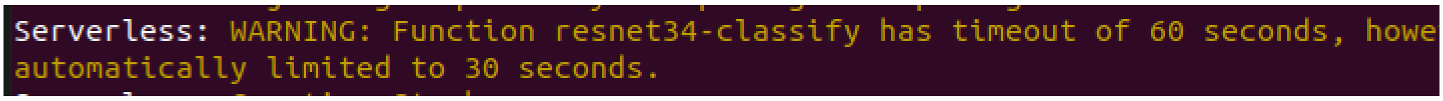
basic-resnet-pytorch-dev-detect_damage

This function belongs to an application. Click here to manage it.


Configuration
Permissions
Monitoring
Designer
Go back to application basic-resnet-pytorch-dev
basic-resnet-pytorch-dev-detect_damage
Layers
(0)
API Gateway
Add trigger

TESTING OUR DEPLOYMENT

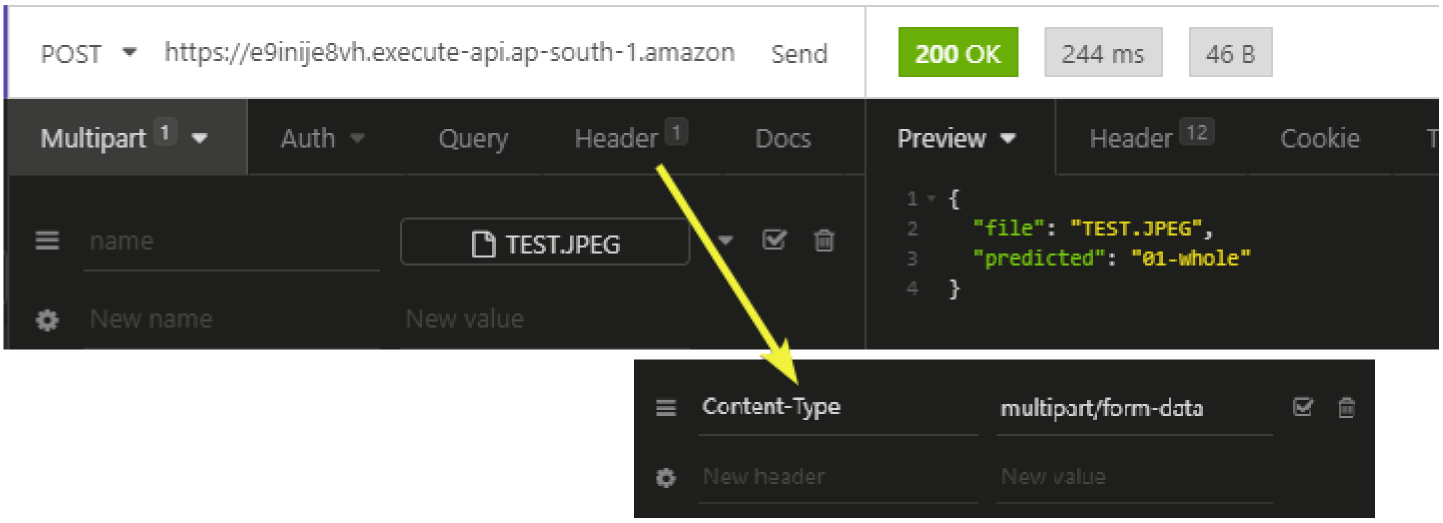
Even if we took care of everything, the first deployment won't show results.



```
Serverless: WARNING: Function resnet34-classify has timeout of 60 seconds, howe
automatically limited to 30 seconds.
```

What you are seeing is a COLD start and that takes time, and if it is more than 30 seconds, we won't get the results we are looking for.

But if you test the second time, you'll get the result (warm start) and it would run must faster as well. Install **Insomnia** (https://insomnia.rest/download/) and follow the steps on the right. Results are shown on the left!


POST https://e9inije8vh.execute-api.ap-south-1.amazon Send
200 OK 244 ms 46 B

Multipart 1 | Auth | Query | Header 1 | Docs

name — TEST.JPEG

New name — New value

Preview | Header 12 | Cookie

```
{
  "file": "TEST.JPEG",
  "predicted": "01-whole"
}
```

Content-Type — multipart/form-data

New header — New value

And we are done, and it took 244ms!! 😅

ASSIGNMENT 1

1. Do the same, but now for **MobileNet V2 (https://pytorch.org/hub/pytorch_vision_mobilenet_v2/)** model.
2. Upload your code to Github and add a screenshot as shown below confirming that you were able to classify **THIS (https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2019/12/03202400/Yellow-Labrador-Retriever.jpg)** image on your Lambda Deployment. I am looking for your Insomnia/etc screenshot like the one shown above:
3. Share the link to your GitHub. **Your GitHub must be a public repo.**

Session Video: