Raspberry PI - Weather Station

User Manual - AWS Version

December 3, 2018

"Patience is a key element of success"

Joynal Abedin

Version History

Date	Version	Author	Comments
12/3/2018	v1.0	Joynal Abedin	Initial Formatting
12/5/2018	v1.1.1	Joynal Abedin	Completed Prerequisite, SensorData
12/5/2018	v1.1.2	Joynal Abedin	Update SensorData, Include Disclaimer
12/6/2018	v1.1.3	Joynal Abedin	Completed Verify-API, Updated SensorData, Updated Disclaimer
12/6/2018	v1.1.4	Joynal Abedin	Updated AWS RDS , Docker
12/7/2018	v1.1.5	Joynal Abedin	AWS Deployment, Elastic load balancer, AutoScaling group
12/8/2018	v1.1.6	Joynal Abedin	Review and update: SensorData, Verify-API, Elastic load balancer
12/9/2018	v2.0	Joynal Abedin	Completed documentation

Table of Contents

Disclaimer

Use this manual at your own risk and responsibility. I will not be responsible for any AWS Costs or consequences that may result from following this manual. I'd highly encourage you to read the AWS Free Tier policy and user agreement before continuing with this document.

AWS Free Tier Policy: https://aws.amazon.com/free/
User agreement: https://aws.amazon.com/agreement/

Manual usage

The symbol with one vertical stroke which is also known as dollar symbol ($) will represent commands in terminal. So, if you see a line followed by $, it means you have to open up a terminal and execute that line of command.

This guide will also assume that you have sufficient understanding of your own system and how to work around terminal which you have learned in 4420 which is the prerequisite of this course.

Prerequisite

System Requirement

I was using macOS Sierra while writing this document. This document may or may not work accurately with other operating system.

If you could get the following item on your system, you may be able to follow along the instructions without any issues.

Dependencies:

1. SSH
2. Docker version 18.03.1-ce, build 9ee9f40 or newer version
3. Account with AWS (free tier will work)
4. AWS CLI v1.16.67+
5. Python 3.6+

Clone the Repository

```
$ git https://github.com/Joy57/WSU-RaspberryPi-WeatherStation-Final.git
```

Get an AWS Account

If you already have an account with AWS, skip this step

1. Visit the Amazon Web Service at https://aws.amazon.com
2. Choose Create an AWS Account.

   Note: If you've signed in to AWS recently, it might say Sign In to the Console. Otherwise it should say Create an AWS Account.

   If the option “Create a new AWS Account” isn’t displayed, first choose to Sign in to a different account, and select “Create a new AWS Account”.

3. Type the requested account information, and then choose Continue.
4. Choose Personal .
5. Type the requested personal information.
6. Read and check the AWS Customer Agreement.
7. Choose Create Account and Continue
8. Add a payment method (if you cross the free tier limit, AWS may start charging you. Here’s more detail about free tier https://aws.amazon.com/free)

9. Verify your phone number
10. Choose the Basic support plan
11. If all the above steps went well, congratulations! Now, you have an AWS Account.

Install AWS CLI on your machine

Requirements

- Python 2 version 2.6.5+ or Python 3 version 3.3+
- Windows, Linux, macOS or Unix

Install the AWS CLI on macOS/Linux

Step 1: Install Python 3

*If you already have it installed, skip this step

```
$ python --version
```

or

```
$ python3 --version
```

If you don't have python installed. Get it from here https://www.python.org/downloads/

Once python is being installed, run the following:

```
$ pip --version
```

If you already have pip, skip the below step for pip installation.

Step 2: Install pip:

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
```

```
$ python get-pip.py --user
```

Add the executable path to your PATH variable: ~/.local/bin

Find your shell profile script.
Add an export command to your profile script

```
$ export PATH=~/.local/bin:$PATH
```

You can now close the terminal and re-open it again to load the profile to your session. Or you can simply do `$ source ~/.bash_profile`

Move on to the next step if you have pip installed successfully.

Step 3: Install AWS CLI

Depend on your system, if it's called pip or pip3, change the below command accordingly.
For me it's pip3.

```
$ pip3  install awscli
```

```
$ aws --version
```

Output: AWS CLI [version number]

Free Tips: If you don't see a version number outputted, then you'd need to add it to your command line path.

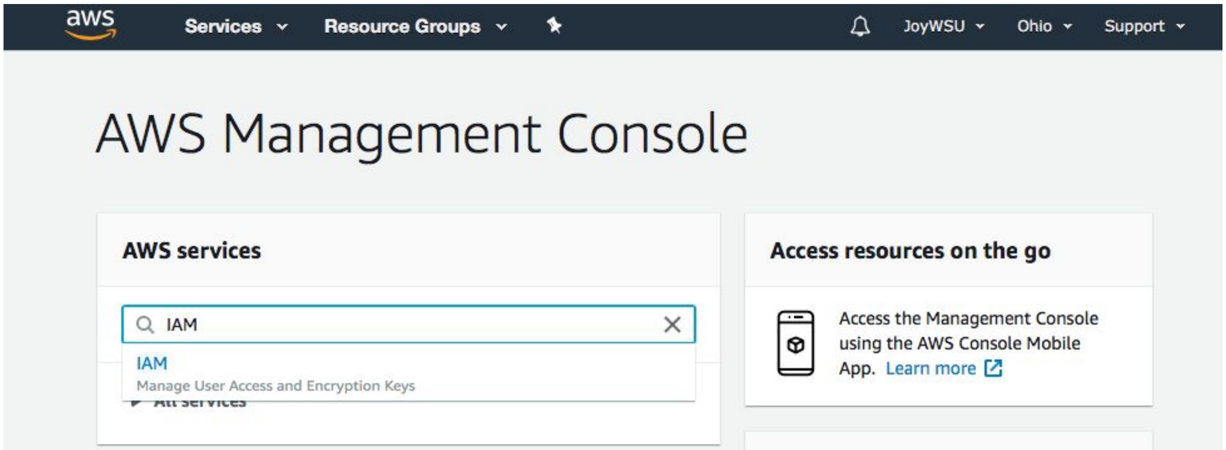
Install the AWS CLI on Windows

MSI Installer for Windows people here:
https://docs.aws.amazon.com/cli/latest/userguide/awscli-install-windows.html

Create an IAM User

Go to https://aws.amazon.com/console
Click on "Sign In to the Console"
Enter your email address
Click next
Enter your password
Click on "Sign in"

From the Home page
Search “IAM” in AWS services and click on the result where it says “IAM”



AWS Management Console

AWS services

IAM

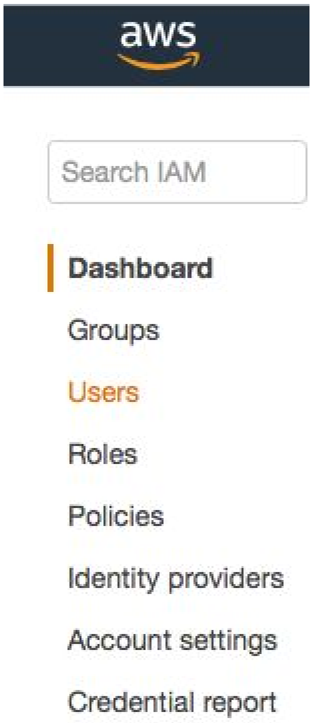
IAM
Manage User Access and Encryption Keys

All services

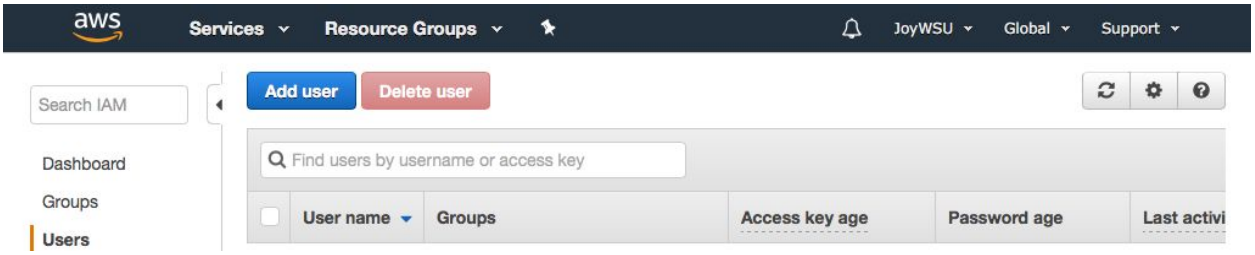
Access resources on the go

Access the Management Console using the AWS Console Mobile App. Learn more

From the left menu choose “Users”



Click on “Add user”



Set user details:

Type the username “AWS-ADMIN-WSU”

Select AWS access type:

Access type: check “Programmatic access”
On the bottom right click “Next: Permissions”

Set permissions

Select **Attach existing policies directly**
In the Filter policies, search “AdministratorAccess”
Select the Policy Name that says “AdministratorAccess”

On the bottom right click “Next: Tags”
On the bottom right click “Next: Review”
On the bottom right click “Create User”

Click on Download.csv button
Note down the value from the column “Access key ID” and “Secret access key”

Configure your CLI on terminal

```
$ aws configure --profile joyWSU

AWS Access Key ID [None]: your_access_key_id
AWS Secret Access Key [None]: your_secret_access_key
Default region name [None]: us-east-1
Default output format [None]:
```

```
$ PROFILE=joyWSU
```

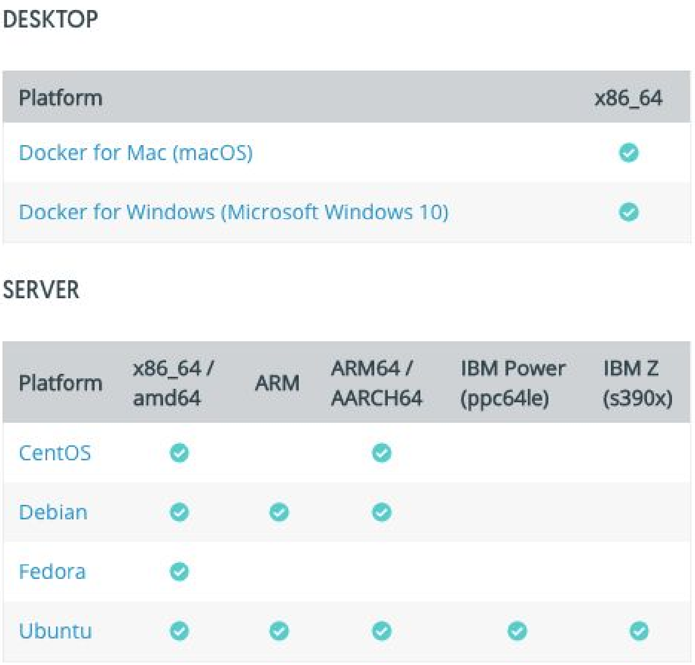
Create S3 Bucket

```
$ aws --profile $PROFILE s3api create-bucket --bucket
raspberry-pi-wsu-senior --region us-east-1
```

```
Response:

{ "Location": "/raspberry-pi-wsu-senior" }
```

Install Docker

Go to https://docs.docker.com/install/
And select the package you want to install (Depends on which operating system you have). If you have a macOS, I'd highly recommend you to use it for this guide.



DESKTOP

Platform	x86_64
Docker for Mac (macOS)	✓
Docker for Windows (Microsoft Windows 10)	✓

SERVER

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64	IBM Power (ppc64le)	IBM Z (s390x)
CentOS	✓		✓		
Debian	✓	✓	✓		
Fedora	✓				
Ubuntu	✓	✓	✓	✓	✓

Open a DockerHub Account

Create a dockerhub account at https://hub.docker.com/

AWS RDS - MySQL

As of today December 5th, 2018, AWS RDS only allows dbs encryption on t2.small (lowest cost). No free tier option available for encryption. So, this will cost you money. Check AWS cost usage for exact price (it may change time to time).

Prerequisite: SensorData, Verify-API sections are completed

Create new policy

Make sure your in the policies directory in the deployment_package

```
$ aws --profile $PROFILE iam create-policy --policy-name RDS-Policy
--policy-document file://policies/RDS.json

Response: {
    "Policy": {
        "PolicyName": "RDS-Policy",
        "PolicyId": "ANPAIC24NBSCG6A3T5AS6",
        "Arn": "arn:aws:iam::667671192261:policy/RDS-Policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2018-12-05T17:05:43Z",
        "UpdateDate": "2018-12-05T17:05:43Z"
    }
}
```

From the response, copy the ARN for later usage
For example:
Arn: “arn:aws:iam::667671192261:policy/RDS-policy”

Create IAM Role

```
$ aws --profile $PROFILE iam create-role --role-name WSU-RDS-ROLE
--assume-role-policy-document file://policies/trusty.json
```

```
Response: {
    "Role": {
        "Path": "/",
        "RoleName": "WSU-RDS-ROLE",
        "RoleId": "AROAIFLRPDAESBHCFKORS",
        "Arn": "arn:aws:iam::667671192261:role/WSU-RDS-ROLE",
        "CreateDate": "2018-12-05T17:09:13Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

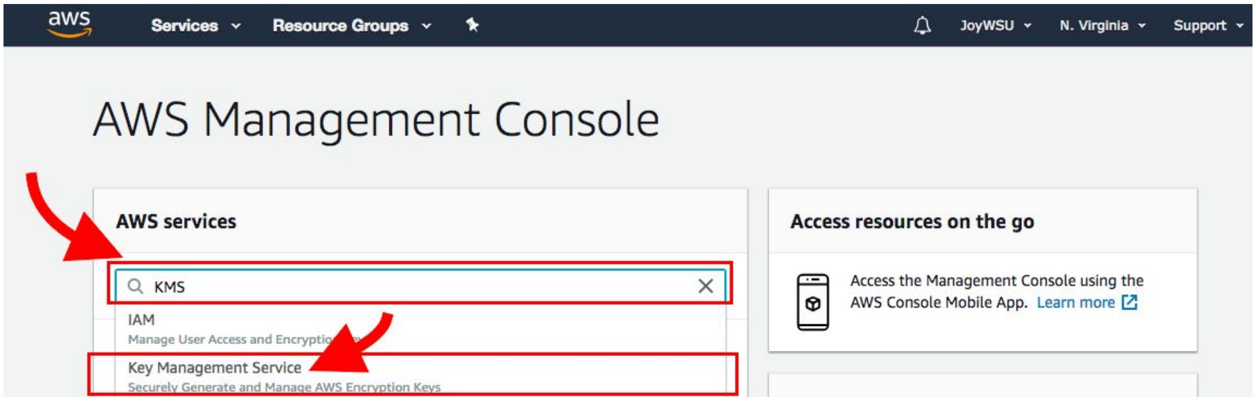
Note: Make sure to replace the red highlighted portion with the appropriate arn name from the previous steps.

Attach the policy to the role

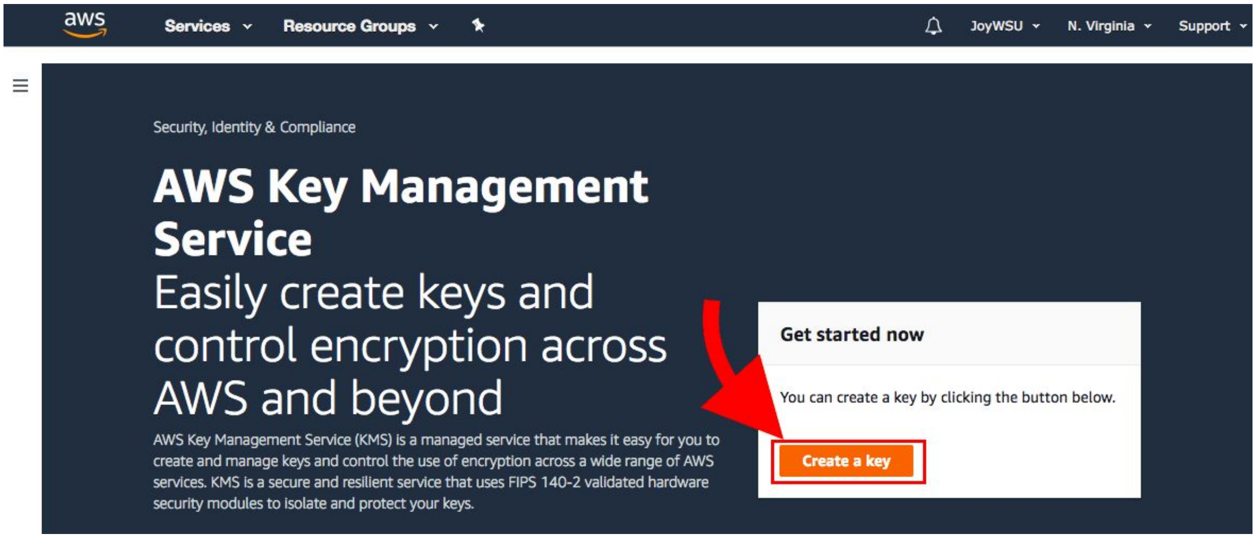
```
$ echo $ACCOUNT
1234567891
# Replace the YOUR_ACCOUNT_ID with your the above output
$ aws --profile $PROFILE iam attach-role-policy --role-name WSU-RDS-ROLE
--policy-arn arn:aws:iam::YOUR_ACCOUNT_ID:policy/RDS-policy
```

Create KMS Key

Search for ‘KMS’


aws
Services
Resource Groups
JoyWSU
N. Virginia
Support
AWS Management Console
AWS services
KMS
IAM
Manage User Access and Encryption Keys
Key Management Service
Securely Generate and Manage AWS Encryption Keys
Access resources on the go
Access the Management Console using the AWS Console Mobile App. Learn more

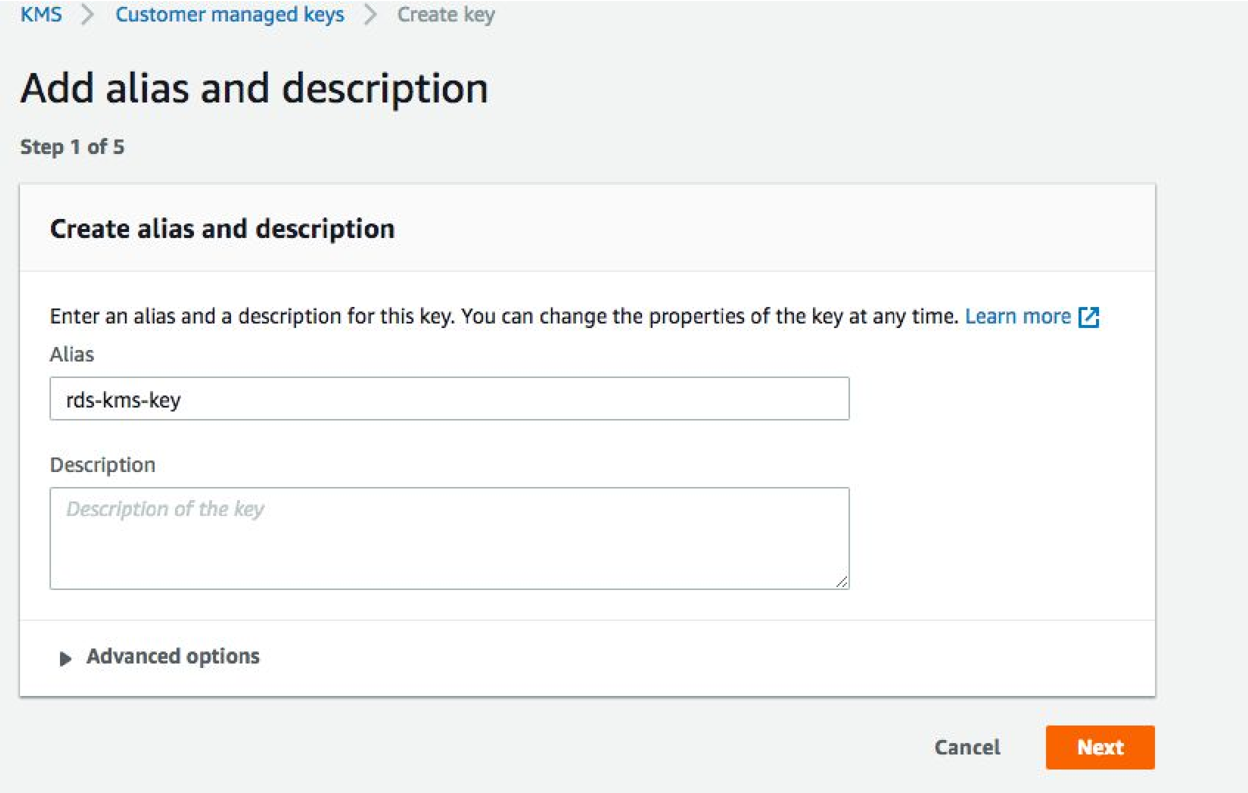
Click on Create a key


aws
Services
Resource Groups
JoyWSU
N. Virginia
Support
Security, Identity & Compliance
AWS Key Management Service
Easily create keys and control encryption across AWS and beyond
Get started now
Create a key

AWS Key Management Service (KMS) is a managed service that makes it easy for you to create and manage keys and control the use of encryption across a wide range of AWS services. KMS is a secure and resilient service that uses FIPS 140-2 validated hardware security modules to isolate and protect your keys.

You can create a key by clicking the button below.

Provide a Display name for the key: “rds-kms-key”



Add alias and description

Step 1 of 5

Create alias and description

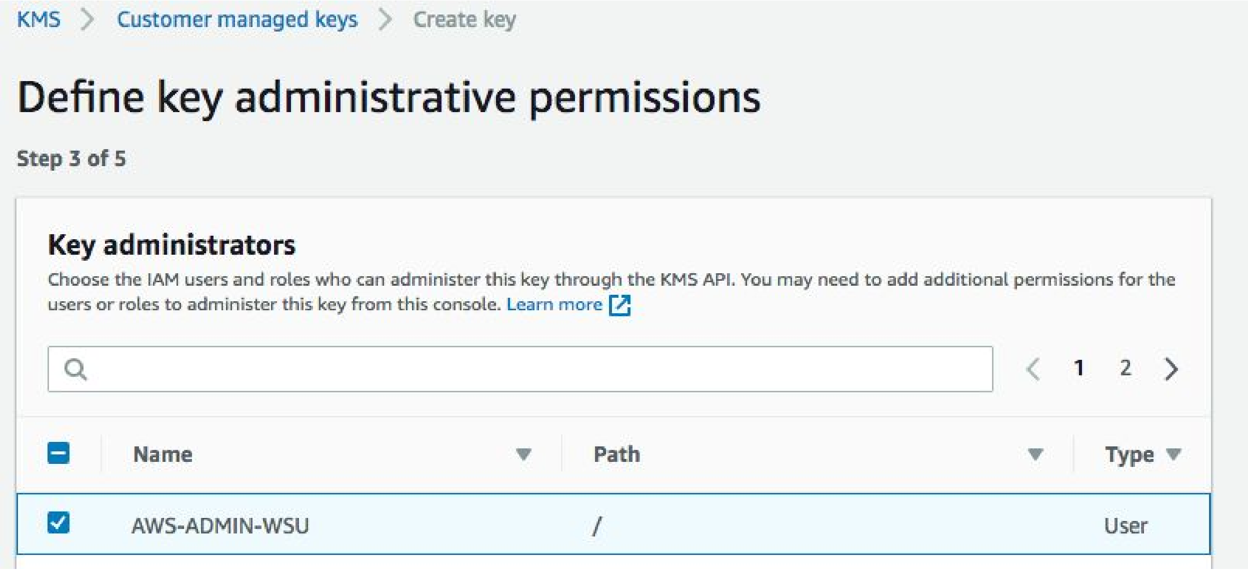
Enter an alias and a description for this key. You can change the properties of the key at any time. Learn more

Alias: rds-kms-key

Description: *Description of the key*

Click on Next.
Add Tags (not required), click on next.

Select ‘AWS-ADMIN-WSU’



Define key administrative permissions

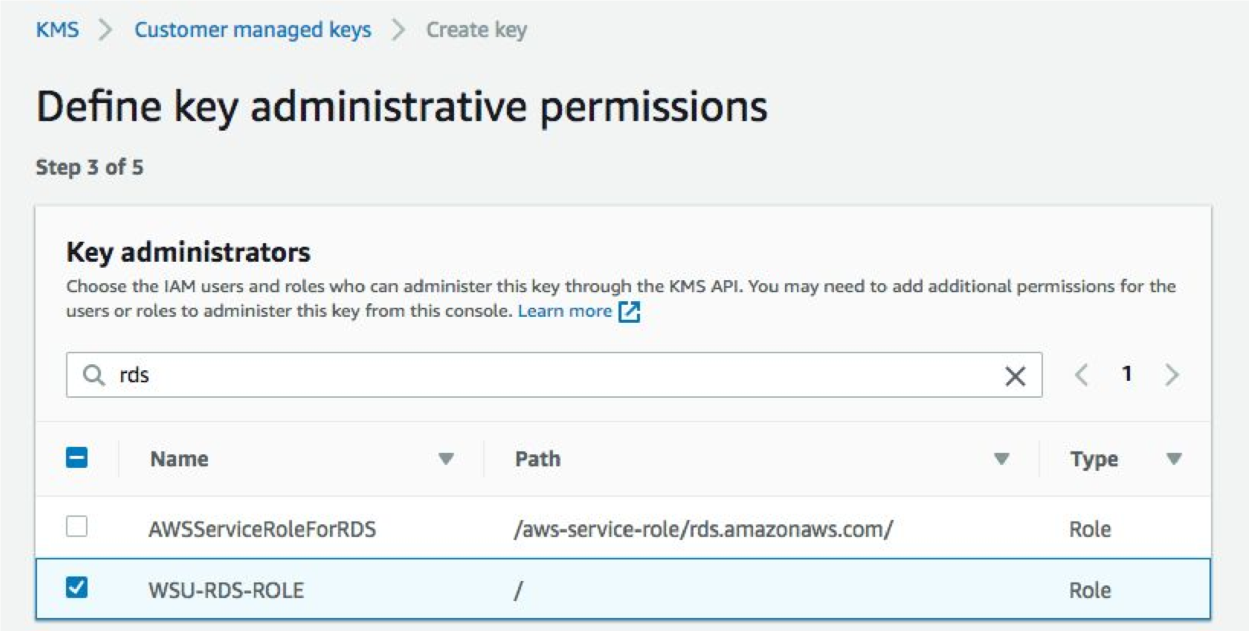
Step 3 of 5

Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. Learn more

☐	Name	Path	Type
☑	AWS-ADMIN-WSU	/	User

And search for “WSU-RDS-ROLE” and select it.



Define key administrative permissions

Step 3 of 5

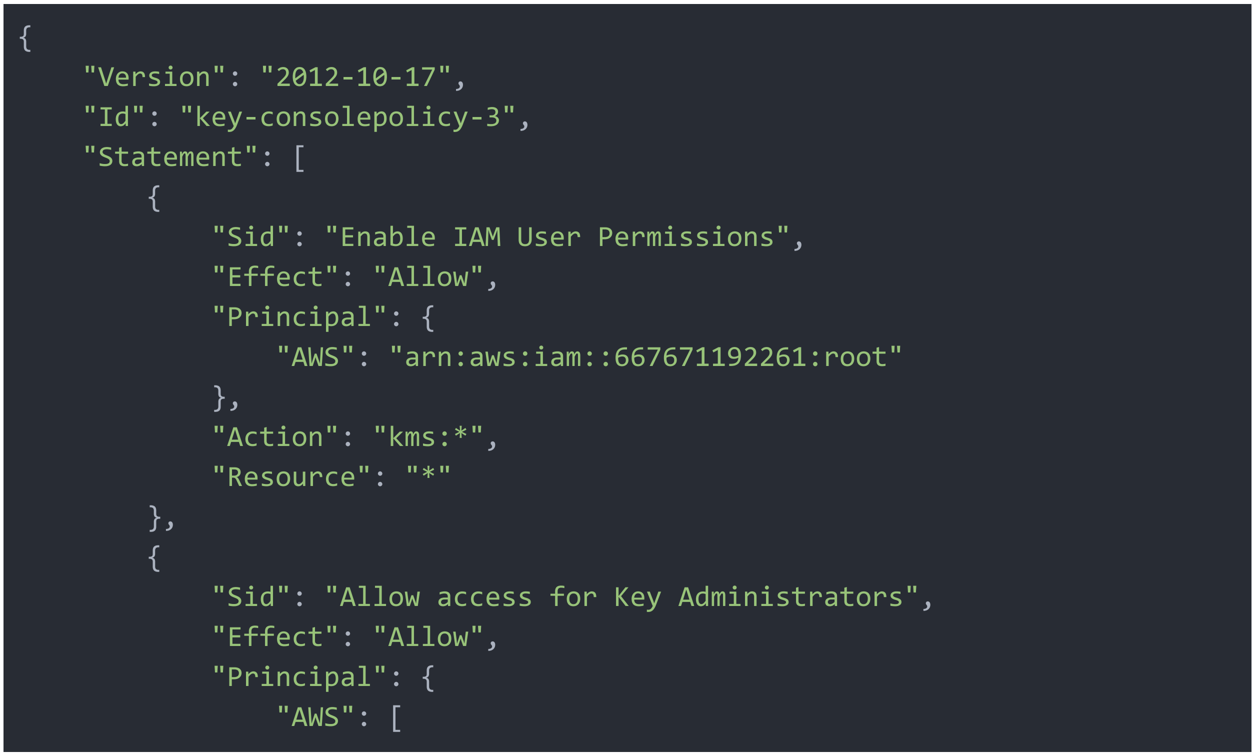
Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. Learn more

☐	Name	Path	Type
☐	AWSServiceRoleForRDS	/aws-service-role/rds.amazonaws.com/	Role
☑	WSU-RDS-ROLE	/	Role

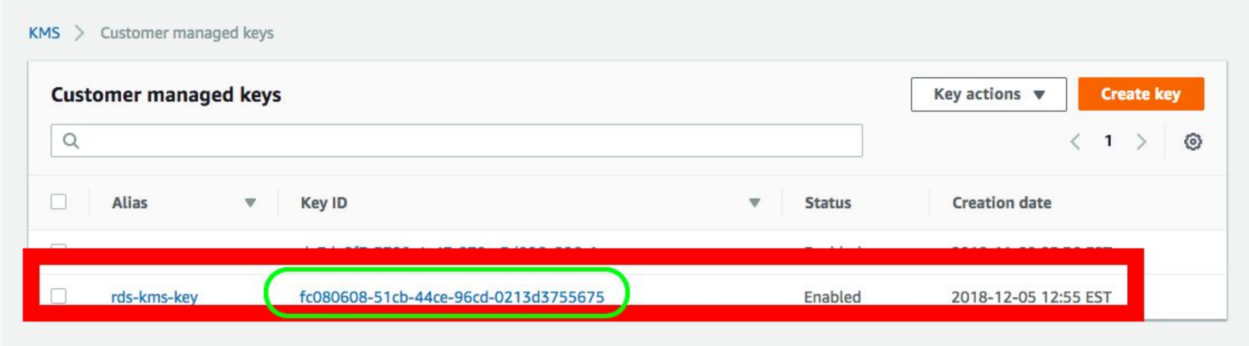
Click on “Next”
Select the “AWS-ADMIN-WSU” and “WSU-RDS-ROLE” for Define key usage permissions as well. And click on “Next”

Review the key policy and click on “Finish”



```
{
    "Version": "2012-10-17",
    "Id": "key-consolepolicy-3",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::667671192261:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow access for Key Administrators",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::667671192261:user/AWS-ADMIN-WSU",
                    "arn:aws:iam::667671192261:role/WSU-RDS-ROLE"
                ]
            },
            "Action": [
                "kms:Create*",
                "kms:Describe*",
                "kms:Enable*",
                "kms:List*",
                "kms:Put*",
                "kms:Update*",
                "kms:Revoke*",
                "kms:Disable*",
                "kms:Get*",
                "kms:Delete*",
                "kms:TagResource",
                "kms:UntagResource",
                "kms:ScheduleKeyDeletion",
                "kms:CancelKeyDeletion"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::667671192261:role/WSU-RDS-ROLE"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::667671192261:role/WSU-RDS-ROLE"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

Now copy the Key ID into the clipboard



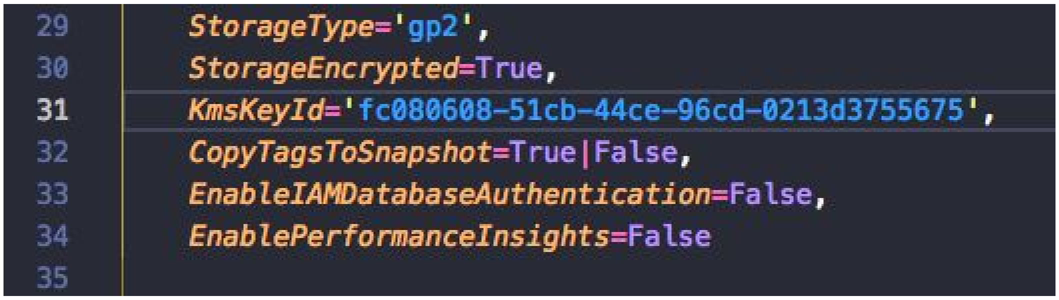
	Alias	Key ID	Status	Creation date
□	rds-kms-key	fc080608-51cb-44ce-96cd-0213d3755675	Enabled	2018-12-05 12:55 EST

In my case, it is `fc080608-51cb-44ce-96cd-0213d3755675`

Make sure you are in the "WSU-RaspberryPi-WeatherStation-Final"

```
$ cd lambda
```

Now open the file "rds_function.py" and replace the KmsKeyId:



```
        StorageType='gp2',
        StorageEncrypted=True,
        KmsKeyId='fc080608-51cb-44ce-96cd-0213d3755675',
        CopyTagsToSnapshot=True|False,
        EnableIAMDatabaseAuthentication=False,
        EnablePerformanceInsights=False

```

Now, compress/zip the same file “rds_function.py”. After compress/zip, name the file “aws_rds.zip”

```
$ aws --profile $PROFILE s3 cp lambda/aws_rds.zip s3://raspberry-pi-wsu-senior
```

Create new Lambda function

```
$ echo $ACCOUNT
1234567891
# Replace the YOUR_ACCOUNT_ID with your the above output
$ aws --profile $PROFILE lambda create-function --function-name AWSRDS
--runtime python3.6 --timeout 200 --role
arn:aws:iam::YOUR_ACCOUNT_ID:role/WSU-RDS-ROLE --handler
rds_function.lambda_handler --code
S3Bucket=raspberry-pi-wsu-senior,S3Key=aws_rds.zip

 Response: {
    "FunctionName": "AWSRDS",
    "FunctionArn": "arn:aws:lambda:us-east-1:667671192261:function:AWSRDS",
    "Runtime": "python3.6",
    "Role": "arn:aws:iam::667671192261:role/WSU-RDS-ROLE",
    "Handler": "rds_function.lambda_handler",
    "CodeSize": 700,
    "Description": "",
    "Timeout": 200,
    "MemorySize": 128,
    "LastModified": "2018-12-05T17:18:04.766+0000",
    "CodeSha256": "kblpiu2FFl3eCYdSaDquQb05MfcVW3D4IiDsf+0ywys=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "731a6cad-99f6-45fa-bc95-8786ec3309d7"
}
```

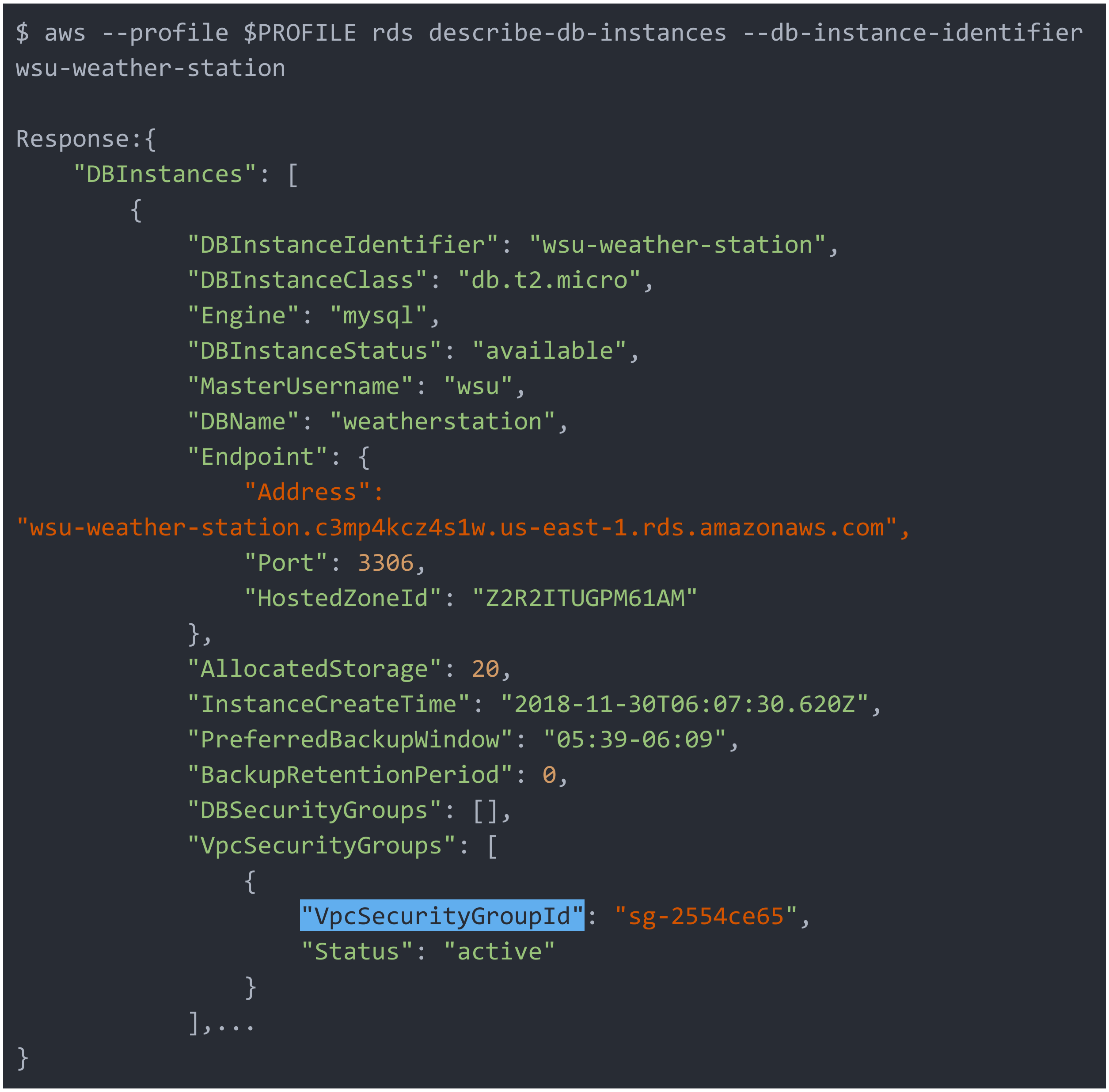
Invoke the lambda function

Copy the output from the echo $ACCOUNT and save it in a separate place as well for late usage. Next time, when you need it again, I’ll refer it as YOUR_ACCOUNT_ID

```
$ echo $ACCOUNT
1234567891
# Replace the YOUR_ACCOUNT_ID with your the above output
```

```
$ aws --profile $PROFILE lambda invoke --function-name
arn:aws:lambda:us-east-1:YOUR_ACCOUNT_ID:function:AWSRDS outfile.txt
```

After executing the above command, please wait about 5 minutes. And then proceed onto the next step.

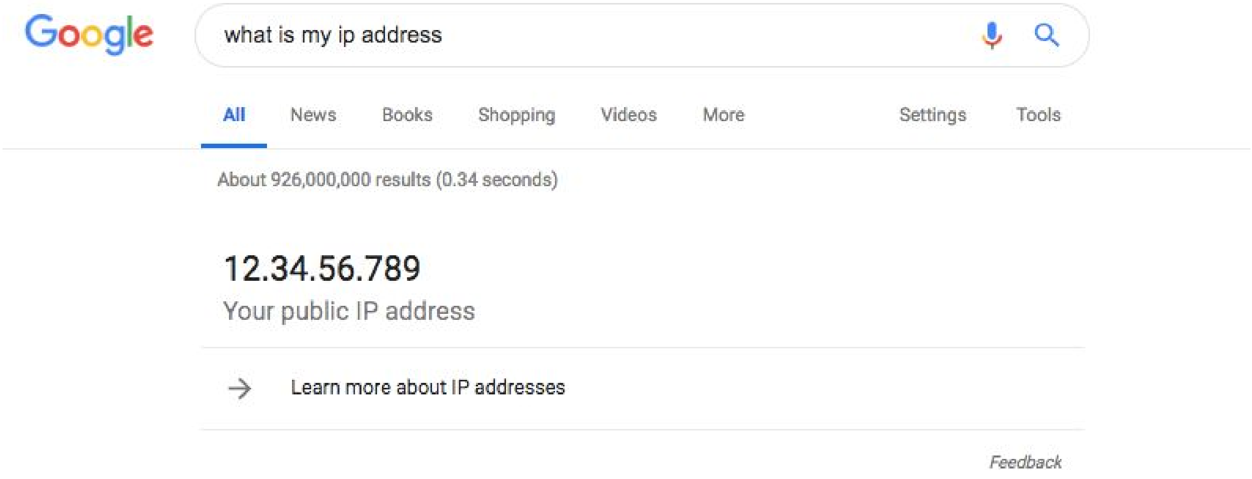


```
$ aws --profile $PROFILE rds describe-db-instances --db-instance-identifier
wsu-weather-station

Response:{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "wsu-weather-station",
            "DBInstanceClass": "db.t2.micro",
            "Engine": "mysql",
            "DBInstanceStatus": "available",
            "MasterUsername": "wsu",
            "DBName": "weatherstation",
            "Endpoint": {
                "Address":
"wsu-weather-station.c3mp4kcz4s1w.us-east-1.rds.amazonaws.com",
                "Port": 3306,
                "HostedZoneId": "Z2R2ITUGPM61AM"
            },
            "AllocatedStorage": 20,
            "InstanceCreateTime": "2018-11-30T06:07:30.620Z",
            "PreferredBackupWindow": "05:39-06:09",
            "BackupRetentionPeriod": 0,
            "DBSecurityGroups": [],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-2554ce65",
                    "Status": "active"
                }
            ],...
}
```

Copy the Address value from the above step and assign it to ENDPOINT. Copy the VpcSecurityGroupId and assign it to VPCSG. Also, keep this db ENDPOINT saved somewhere else, we'll use it later to update AWS Lambda.

```
$ ENDPOINT=wsu-weather-station.c3mp4kcz4s1w.us-east-1.rds.amazonaws.com
$ VPCSG=sg-2554ce65
```

Find your public ip address. You can do a google search “What is my ip address”.



what is my ip address

About 926,000,000 results (0.34 seconds)

12.34.56.789

Your public IP address

Learn more about IP addresses

Do not forget the /32 at the end of your ip address

```
$ MYIP=YOUR_PUBLIC_IP_ADDRESS/32
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-id
$VPCSG --protocol tcp --port 3306  --cidr $MYIP
```

SensorData

Upload code to S3 Bucket

Make sure you are in the folder “WSU-RaspberryPi-WeatherStation-Final”

```
cd lambda/sensordata/
```

Open the file called rds_config.py.
Change the db_host value with the new db ENDPOINT from previous steps.

```
db_host ="old_endpoint"
```

Now make sure you are in the root folder “WSU-RaspberryPi-WeatherStation-Final” before executing below scripts.

```
$ aws --profile $PROFILE s3 cp lambda/sensordata.zip
s3://raspberry-pi-wsu-senior
```

Create new policy

Make sure your in the policies directory in the deployment_package

```
$ aws --profile $PROFILE iam create-policy --policy-name sensordata-policy
--policy-document file://policies/sensordata.json
```

```
Response: {
    "Policy": {
        "PolicyName": "sensordata-policy",
        "PolicyId": "ANPAJHNSZ2KAFXC6VLEMK",
        "Arn": "arn:aws:iam::667671192261:policy/sensordata-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2018-12-04T03:21:03Z",
        "UpdateDate": "2018-12-04T03:21:03Z"
    }
}
```

From the response, copy the ARN for later usage
For example:
Arn: “arn:aws:iam::667671192261:policy/sensordata-policy”

```
$ POLICYARN=arn:aws:iam::667671192261:policy/sensordata-policy
# COPY the numbers from above and assign it to ACCOUNT
$ ACCOUNT=667671192261
```

Create IAM Role

```
$ aws --profile $PROFILE iam create-role --role-name WSU-SENSOR-ROLE
--assume-role-policy-document file://policies/trusty.json


---------------------------------------------------------------------------
Response: {
    "Role": {
        "Path": "/",
        "RoleName": "WSU-SENSOR-ROLE",
        "RoleId": "AROAI2K2XLICPSCLLS376",
        "Arn": "arn:aws:iam::667671192261:role/WSU-SENSOR-ROLE",
        "CreateDate": "2018-12-04T03:26:31Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Copy the **ARN** and **Role** name for later usage

For example:
“RoleName”: “WSU-SENSOR-ROLE”
“Arn”: “arn:aws:iam::667671192261:role/WSU-SENSOR-ROLE”

```
$ SENSORROLE=WSU-SENSOR-ROLE
$ SENSORARN=arn:aws:iam::667671192261:role/WSU-SENSOR-ROLE
```

Attach the policy to the role

```
$ aws --profile $PROFILE iam attach-role-policy --role-name $SENSORROLE
--policy-arn $POLICYARN
```

Get VPC info

Make note of the VpcId from the response

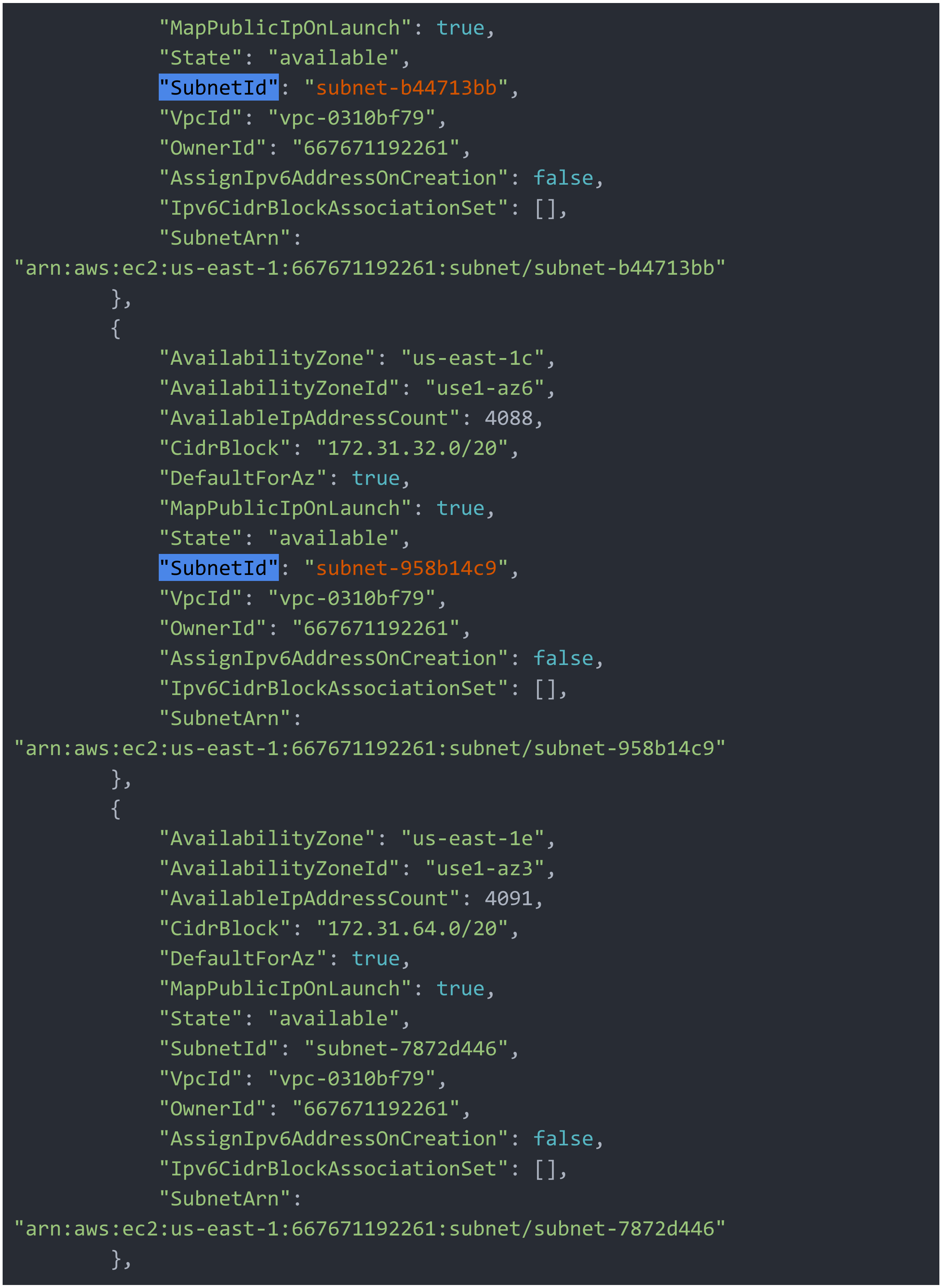
```
$ aws --profile $PROFILE ec2 describe-vpcs --filters
```

```
Name=isDefault,Values=true

Response: {
    "Vpcs": [
        {
            "CidrBlock": "172.31.0.0/16",
            "DhcpOptionsId": "dopt-3ed09845",
            "State": "available",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "InstanceTenancy": "default",
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-a8fe45c4",
                    "CidrBlock": "172.31.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": true
        }
    ]
}
```

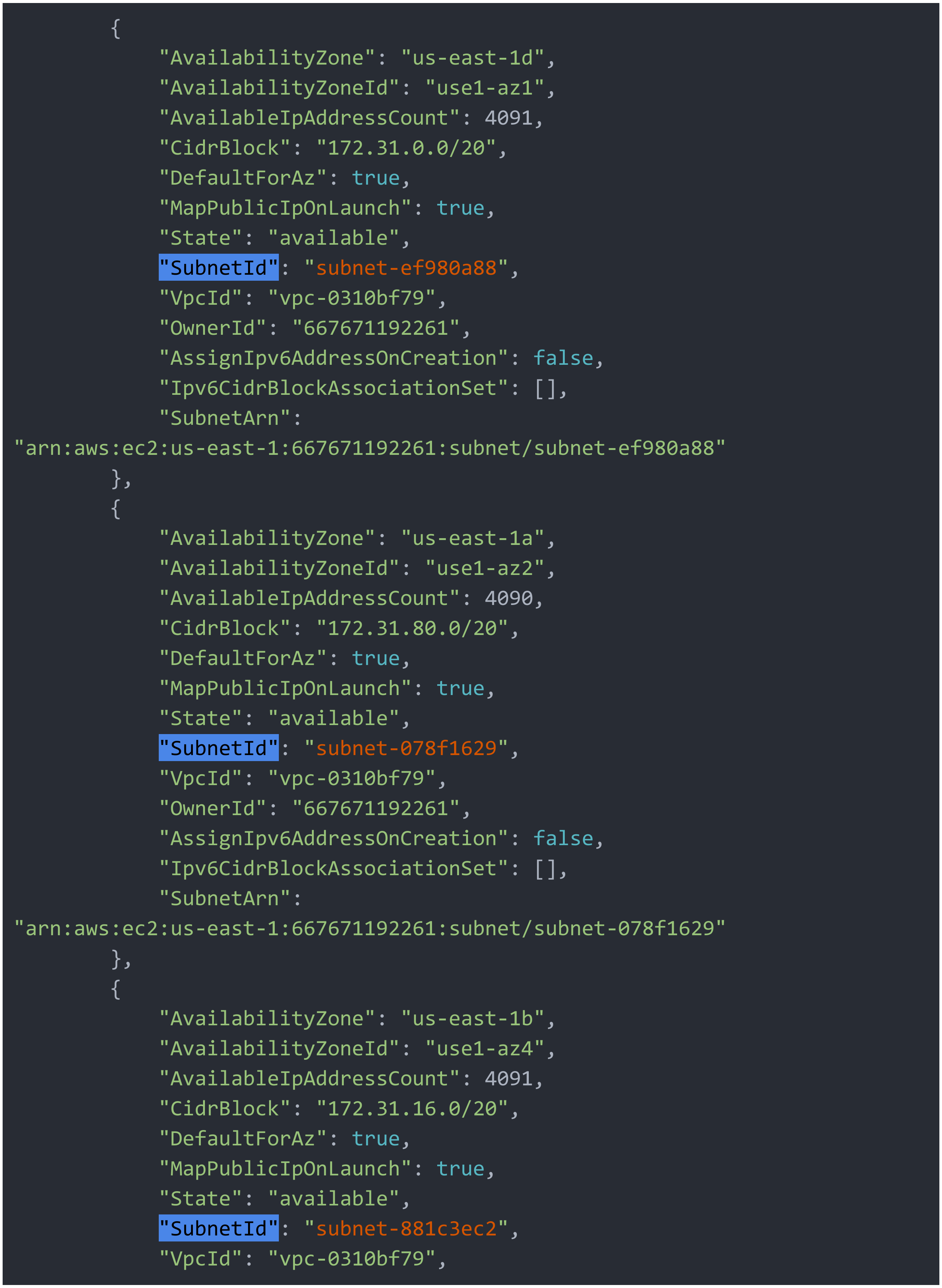
Describe VPC subnets

Make note of all of the SubnetId from the response. You'll use it later. They're highlighted with red font followed by "SubnetId" parameter

```
$ VPC=vpc-0310bf79
$ aws --profile $PROFILE ec2 describe-subnets --filters
Name=vpc-id,Values=$VPC Name=default-for-az,Values=true

Response: {
    "Subnets": [
        {
            "AvailabilityZone": "us-east-1f",
            "AvailabilityZoneId": "use1-az5",
            "AvailableIpAddressCount": 4088,
            "CidrBlock": "172.31.48.0/20",
            "DefaultForAz": true,
```



```
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-b44713bb",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-b44713bb"
        },
        {
            "AvailabilityZone": "us-east-1c",
            "AvailabilityZoneId": "use1-az6",
            "AvailableIpAddressCount": 4088,
            "CidrBlock": "172.31.32.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-958b14c9",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-958b14c9"
        },
        {
            "AvailabilityZone": "us-east-1e",
            "AvailabilityZoneId": "use1-az3",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.64.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-7872d446",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-7872d446"
        },
```



```
        {
            "AvailabilityZone": "us-east-1d",
            "AvailabilityZoneId": "use1-az1",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.0.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-ef980a88",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-ef980a88"
        },
        {
            "AvailabilityZone": "us-east-1a",
            "AvailabilityZoneId": "use1-az2",
            "AvailableIpAddressCount": 4090,
            "CidrBlock": "172.31.80.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-078f1629",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-078f1629"
        },
        {
            "AvailabilityZone": "us-east-1b",
            "AvailabilityZoneId": "use1-az4",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.16.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-881c3ec2",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-881c3ec2"
        }
    ]
}
```

Get the Security Group ID

Make note of the “GroupID” parameter’s value.

```
$ aws --profile $PROFILE ec2 describe-security-groups --filters
Name=vpc-id,Values=$VPC --group-names default

Response: {
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": [
                        {
                            "GroupId": "sg-6c47e324",
                            "UserId": "804994069721"
                        }
                    ]
                }
            ],
            "OwnerId": "804994069721",
            "GroupId": "sg-6c47e324",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": []
                }
            ],
            "VpcId": "vpc-ca8ea2b1"
        }
    ]
}
```

Create new Lambda function

Make sure to replace the SubnetIds and the SecurityGroupIds. They're highlighted in red.

```
$ echo $ACCOUNT
804994069721
# this is your account id. Copy it and replace with the text that says
YOUR_ACCOUNT_ID
$ aws --profile $PROFILE lambda create-function --function-name sensordata
--runtime python3.6 --role
arn:aws:iam::YOUR_ACCOUNT_ID:role/WSU-SENSOR-ROLE --handler
lambda_function.handler --code
S3Bucket=raspberry-pi-wsu-senior,S3Key=sensordata.zip --vpc-config
SubnetIds=subnet-b44713bb,subnet-958b14c9,subnet-ef980a88,subnet-078f1629,s
ubnet-881c3ec2,SecurityGroupIds=sg-2554ce65

Response: {
    "FunctionName": "sensordata",
    "FunctionArn":
"arn:aws:lambda:us-east-1:667671192261:function:sensordata2",
    "Runtime": "python3.6",
    "Role": "arn:aws:iam::667671192261:role/WSU-SENSOR-ROLE",
    "Handler": "lambda_function.handler",
    "CodeSize": 1984640,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-12-05T19:38:51.311+0000",
    "CodeSha256": "Z1AHTgsmJOQQDttuLZXujdsOwSIg0qdJekuE7tsbGtE=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [
            "subnet-958b14c9",
            "subnet-ef980a88",
            "subnet-078f1629",
            "subnet-881c3ec2",
            "subnet-b44713bb"
        ],
        "SecurityGroupIds": [
            "sg-2554ce65"
        ],
        "VpcId": "vpc-0310bf79"
    },
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "d23924a6-f40f-4d8d-9568-80e03942cf33"
}
```

Setup API Gateway

Create the API

```
$ aws --profile $PROFILE apigateway create-rest-api --name sensordata-API
--endpoint-configuration types=REGIONAL

---------------------------------------------------------------------------
Response: {
    "id": "pkqwe266xg",
    "name": "sensordata-API",
    "createdDate": 1543900164,
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ]
    }
}
```

From the previous step, save the “id” parameter and replace it below with API value. Now, you can get the API root resource ID as well.

```
$ API=pkqwe266xg
$ aws --profile $PROFILE apigateway get-resources --rest-api-id $API

Response: {
    "items": [
        {
            "id": "eu9ur0jp06",
            "path": "/"
        }
    ]
}
```

Copy the id from the response into the clipboard.

Create a Resource in the API

Set the variable ROOT_RESOURCE to the id that you copied from previous step.

```
$ ROOT_RESOURCE=eu9ur0jp06
$ aws --profile $PROFILE apigateway create-resource --rest-api-id $API
--path-part 'sensordata' --parent-id $ROOT_RESOURCE

Response: {
    "id": "txgaza",
    "parentId": "eu9ur0jp06",
    "pathPart": "sensordata",
    "path": "/sensordata"
}
```

Create POST Method on the Resource

```
$ RESOURCE=txgaza
$ aws --profile $PROFILE apigateway put-method --rest-api-id $API
```

```
--resource-id $RESOURCE --http-method POST --authorization-type "NONE"
--api-key-required

Response: {
    "httpMethod": "POST",
    "authorizationType": "NONE",
    "apiKeyRequired": true
}
```

```
$ echo $ACCOUNT
12345678910
# The output is your account id. Copy it and replace with the text that
says YOUR_ACCOUNT_ID
$ aws --profile $PROFILE apigateway put-integration --rest-api-id $API
--resource-id $RESOURCE --http-method POST --type AWS_PROXY
--integration-http-method POST --uri
arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambd
a:us-east-1:YOUR_ACCOUNT_ID:function:sensordata/invocations

Response: {
    "type": "AWS_PROXY",
    "httpMethod": "POST",
    "uri":
"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lamb
da:us-east-1:667671192261:function:test1/invocations",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "txgaza",
    "cacheKeyParameters": []
}
```

```
$ aws --profile $PROFILE apigateway put-method-response --rest-api-id $API
--resource-id $RESOURCE --http-method POST --status-code 200
--response-parameters "method.response.header.Access-Control-Allow-Origin=true"
--response-models application/json=Empty

Response: {
    "statusCode": "200",
    "responseParameters": {
        "method.response.header.Access-Control-Allow-Origin": true
    },
    "responseModels": {
        "application/json": "Empty"
    }
}
```

```
$ aws --profile $PROFILE apigateway put-integration-response --rest-api-id
$API --resource-id $RESOURCE --http-method POST --status-code 200
--selection-pattern "" --response-templates '{"application/json":
"{\"json\": \"Empty\"}"}' --response-parameters
'{"method.response.header.Access-Control-Allow-Origin":"'"'*'"'"}'


Response: {
    "statusCode": "200",
    "selectionPattern": "",
    "responseParameters": {
        "method.response.header.Access-Control-Allow-Origin": "'*'"
    },
    "responseTemplates": {
        "application/json": "{\"json\": \"Empty\"}"
    }
}
```

Deploy the API

```
$ aws --profile $PROFILE apigateway create-deployment --rest-api-id $API
--stage-name final --stage-description 'Final Stage' --description
'Deployment for the weather station pi'

Response: {
    "id": "uc7mww",
    "description": "Deployment for the weather station pi",
    "createdDate": 1543986142
}
```

```
$ echo $ACCOUNT
12345678910
```

```
# The output is your account id. Copy it and replace with the text that
says YOUR_ACCOUNT_ID
$ echo $API
Pkqwe266xg
# The output is your API id. Copy it and replace with the text that says
YOUR_API
$ aws --profile $PROFILE lambda add-permission --function-name sensordata
--statement-id apigateway-testt-1 --action lambda:InvokeFunction
--principal apigateway.amazonaws.com --source-arn
"arn:aws:execute-api:us-east-1:YOUR_ACCOUNT_ID:YOUR_API/*/POST/sensordata"


Response: {
    "Statement":
"{\"Sid\":\"apigateway-testt-1\",\"Effect\":\"Allow\",\"Principal\":{\"Serv
ice\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"
Resource\":\"arn:aws:lambda:us-east-1:667671192261:function:test1\",\"Condi
tion\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-1:6676
71192261:8h35no6zsg/*/POST/sensordata\"}}}"
}
```

```
$ aws --profile $PROFILE apigateway create-api-key --name 'PI-API-Key'
--description 'for development' --enabled --stage-keys
restApiId=$API,stageName=final

Response: {
    "id": "a56g1r3mzc",
    "value": "g7gwWjktqX5rfQLQS7tIs2nXmzPYSpem4SH0W5ze",
    "name": "PI-API-Key",
    "description": "for development",
    "enabled": true,
    "createdDate": 1543986612,
    "lastUpdatedDate": 1543986612,
    "stageKeys": [
        "8h35no6zsg/final"
    ]
}
```

Copy the “id” and “value” from the response above and assign it to the below variables.

```
$ KEYID=a56g1r3mzc
$ XAPIKEY=g7gwWjktqX5rfQLQS7tIs2nXmzPYSpem4SH0W5ze
```

Create usage plan

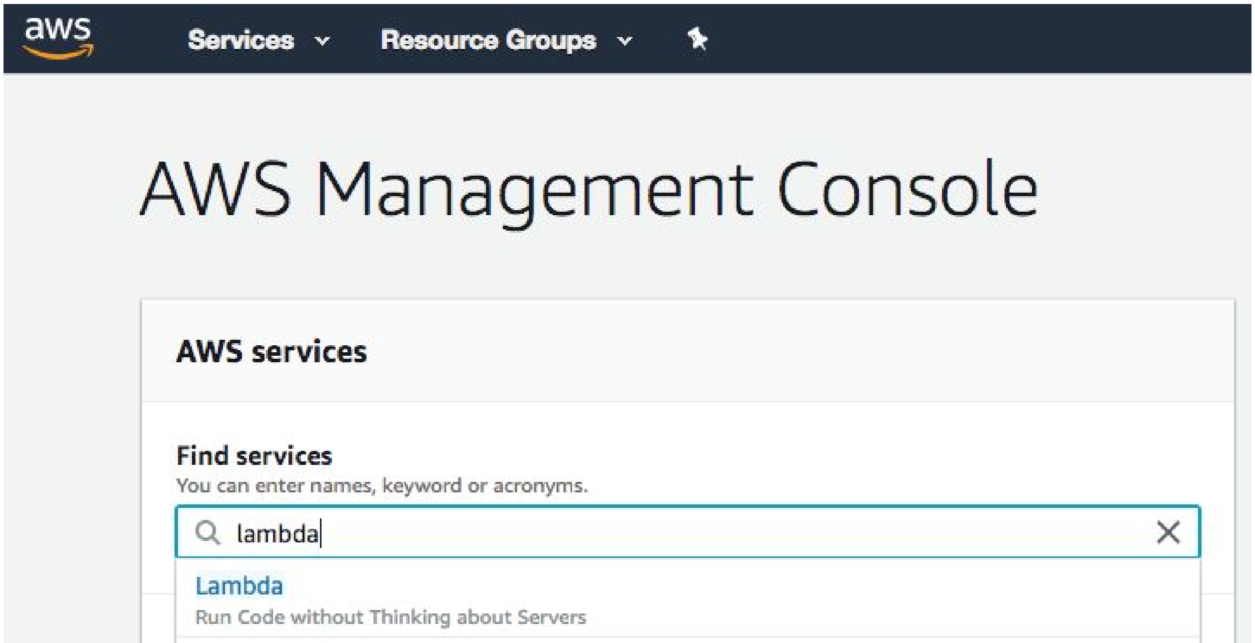
```
$ aws --profile $PROFILE apigateway create-usage-plan --name
"pi-api-key-usagePlan" --api-stages apiId=$API,stage=final --description
"usage plan for pi api key" --throttle burstLimit=10,rateLimit=150 --quota
limit=500,offset=0,period=MONTH

Response:{
    "id": "u9dvxc",
    "name": "pi-api-key-usagePlan",
    "description": "usage plan for pi api key",
    "apiStages": [
        {
            "apiId": "8h35no6zsg",
            "stage": "final"
        }
    ],
    "throttle": {
        "burstLimit": 10,
        "rateLimit": 150.0
    },
    "quota": {
        "limit": 500,
        "offset": 0,
        "period": "MONTH"
    }
}
```

Copy the “id” from the response above and assign it to USAGEID

```
$ USAGEID=u9dvxc

$ aws --profile $PROFILE apigateway create-usage-plan-key --usage-plan-id
$USAGEID --key-type "API_KEY" --key-id $KEYID

Response: {
    "id": "cn02s869kj",
    "type": "API_KEY",
    "name": "PI-API-Key"
}
```

Get your endpoint and x-api-key

Search for lambda



AWS Management Console

AWS services

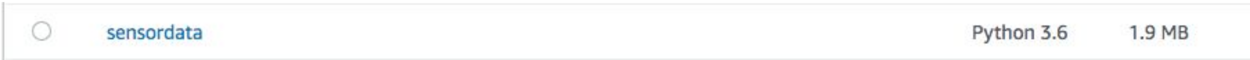
Find services

You can enter names, keyword or acronyms.

lambda

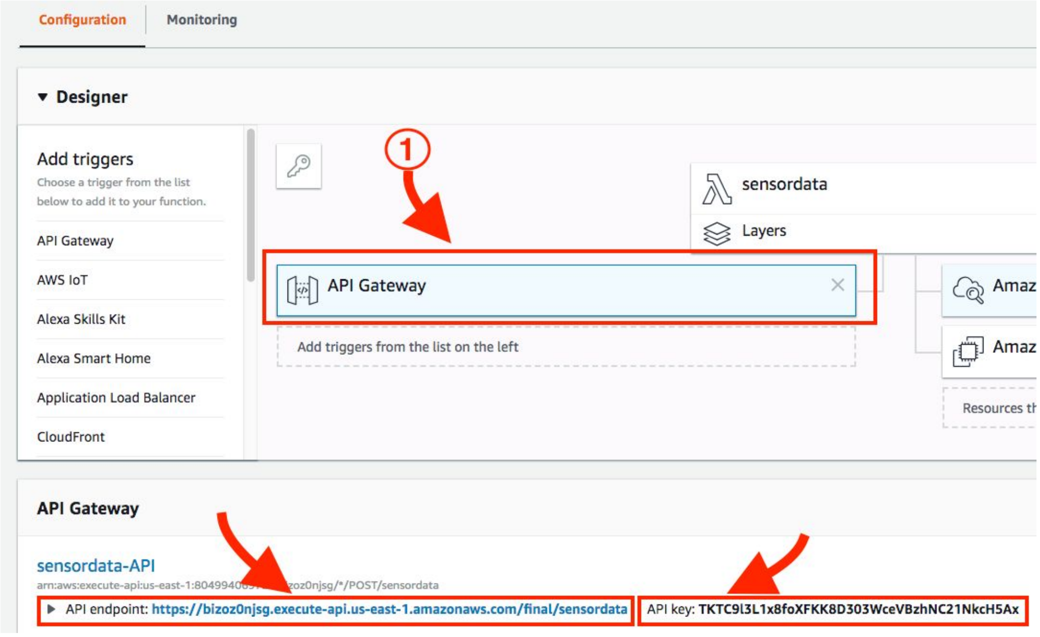
Lambda
Run Code without Thinking about Servers

Click on sensordata



○ sensordata Python 3.6 1.9 MB

Click on API Gateway. Make note of the Api Endpoint, and X-API-KEY (in the picture it's API Key)..



API Gateway

sensordata-API

arn:aws:execute-api:us-east-1:80499406 zoz0njsg/*/POST/sensordata

API endpoint: https://bizoz0njsg.execute-api.us-east-1.amazonaws.com/final/sensordata

API key: TKTC9l3L1x8foXFKK8D303WceVBzhNC21NkcH5Ax

Verify-API

Upload code to S3 Bucket

Make sure you are in the folder “WSU-RaspberryPi-WeatherStation-Final”

```
cd lambda/verifyAPI/
```

Open the file called rds_config.py.
Change the db_host value with the new db ENDPOINT from previous steps.

```
db_host ="old_endpoint"
```

Now make sure you are in the root folder “WSU-RaspberryPi-WeatherStation-Final” before executing below scripts.

```
$ aws s3 cp verifyAPI.zip s3://raspberry-pi-wsu-senior
```

Create new policy

Make sure your in the policies directory in the deployment_package

```
$ aws iam create-policy --policy-name verifyapi-policy --policy-document
file://VerifyAPI.json
```

```
Response: {
    "Policy": {
        "PolicyName": "verifyapi-policy",
        "PolicyId": "ANPAJCETVWRAMPUBBXMXW",
        "Arn": "arn:aws:iam::667671192261:policy/verifyapi-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2018-12-06T02:50:59Z",
        "UpdateDate": "2018-12-06T02:50:59Z"
    }
}
```

From the response, copy the ARN for later usage

For example:
Arn: “arn:aws:iam::667671192261:policy/sensordata-policy”

Create IAM Role

```
$ aws iam create-role --role-name WSU-VERIFY-ROLE
--assume-role-policy-document file://trusty.json

---------------------------------------------------------------------------
Response: {
    "Role": {
        "Path": "/",
        "RoleName": "WSU-VERIFY-ROLE",
        "RoleId": "AROAJBKSEHFMYO5NVKGBC",
        "Arn": "arn:aws:iam::667671192261:role/WSU-VERIFY-ROLE",
        "CreateDate": "2018-12-06T02:51:32Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Copy the **ARN** and **Role** name for later usage

For example:
“RoleName”: “WSU-SENSOR-ROLE”
“Arn”: “arn:aws:iam::667671192261:role/WSU-SENSOR-ROLE”

Note: Make sure to replace the red highlighted portion with the appropriate arn name from the previous steps.

Attach the policy to the role

```
$ aws iam attach-role-policy --role-name WSU-VERIFY-ROLE --policy-arn
arn:aws:iam::667671192261:policy/verifyapi-policy
```

Get VPC info

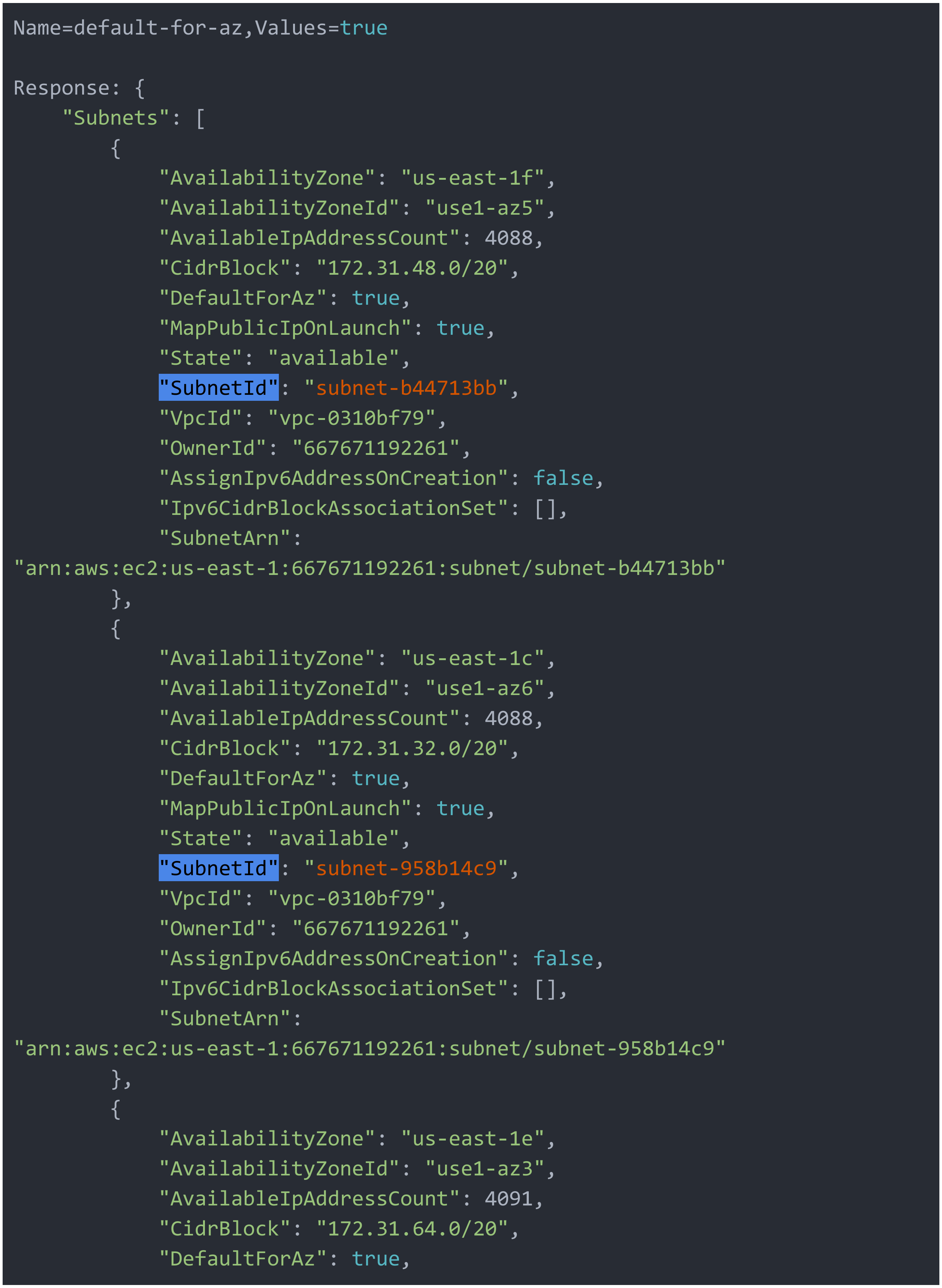
Make note of the VpcId from the response

```
$ aws ec2 describe-vpcs --filters Name=isDefault,Values=true

Response: {
    "Vpcs": [
        {
            "CidrBlock": "172.31.0.0/16",
            "DhcpOptionsId": "dopt-3ed09845",
            "State": "available",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "InstanceTenancy": "default",
            "CidrBlockAssociationSet": [
                {
                    "AssociationId": "vpc-cidr-assoc-a8fe45c4",
                    "CidrBlock": "172.31.0.0/16",
                    "CidrBlockState": {
                        "State": "associated"
                    }
                }
            ],
            "IsDefault": true
        }
    ]
}
```

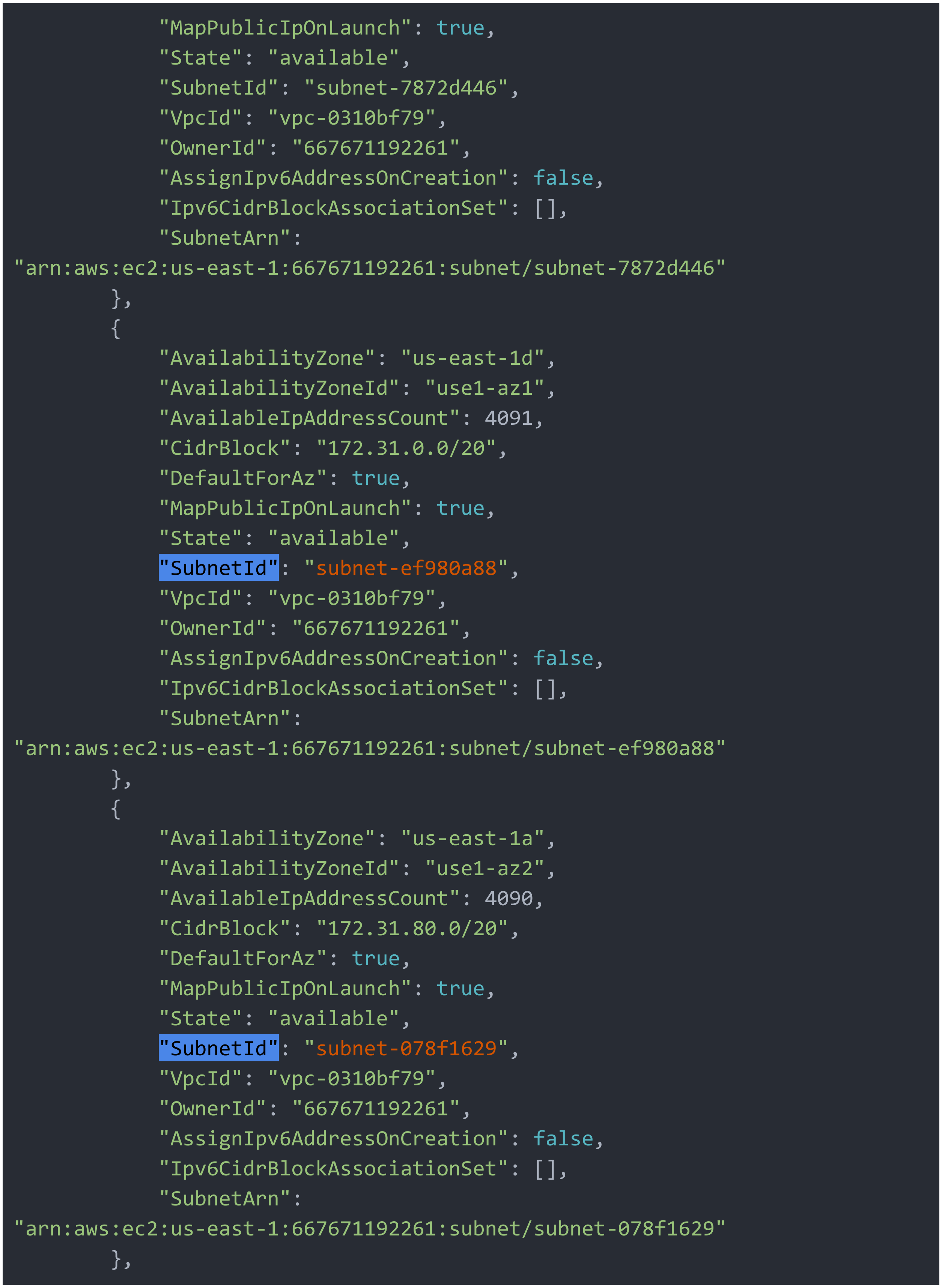
Describe VPC subnets

Make note of all of the SubnetId from the response. You'll use it later. They're highlighted with red font followed by "SubnetId" parameter

```
$ VPC=vpc-0310bf79
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=$VPC
```



```
Name=default-for-az,Values=true

Response: {
    "Subnets": [
        {
            "AvailabilityZone": "us-east-1f",
            "AvailabilityZoneId": "use1-az5",
            "AvailableIpAddressCount": 4088,
            "CidrBlock": "172.31.48.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-b44713bb",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-b44713bb"
        },
        {
            "AvailabilityZone": "us-east-1c",
            "AvailabilityZoneId": "use1-az6",
            "AvailableIpAddressCount": 4088,
            "CidrBlock": "172.31.32.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-958b14c9",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-958b14c9"
        },
        {
            "AvailabilityZone": "us-east-1e",
            "AvailabilityZoneId": "use1-az3",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.64.0/20",
            "DefaultForAz": true,
```



```
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-7872d446",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-7872d446"
        },
        {
            "AvailabilityZone": "us-east-1d",
            "AvailabilityZoneId": "use1-az1",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.0.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-ef980a88",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-ef980a88"
        },
        {
            "AvailabilityZone": "us-east-1a",
            "AvailabilityZoneId": "use1-az2",
            "AvailableIpAddressCount": 4090,
            "CidrBlock": "172.31.80.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-078f1629",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-078f1629"
        },
        {
            "AvailabilityZone": "us-east-1b",
            "AvailabilityZoneId": "use1-az4",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.16.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "State": "available",
            "SubnetId": "subnet-881c3ec2",
            "VpcId": "vpc-0310bf79",
            "OwnerId": "667671192261",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn":
"arn:aws:ec2:us-east-1:667671192261:subnet/subnet-881c3ec2"
        }
    ]
}
```

Get the Security Group ID

Make note of the “GroupID” parameter’s value.

```
$ aws ec2 describe-security-groups --filters Name=vpc-id,Values=$VPC
--group-names default

Response: {
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "FromPort": 3306,
                    "IpProtocol": "tcp",
                    "IpRanges": [
                        {
                            "CidrIp": "172.31.42.214/32",
                            "Description": "ec2-for-AMI"
                        },
                        {
                            "CidrIp": "68.01.01.123/32",
                            "Description": "joy home"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "ToPort": 3306,
                    "UserIdGroupPairs": [
                        {
                            "Description": "ASG Security group",
                            "GroupId": "sg-0a0876f20bed41e27",
                            "UserId": "667671192261"
                        },
                        {
                            "Description": "vpc security",
                            "GroupId": "sg-2554ce65",
                            "UserId": "667671192261"
                        }
                    ]
                }
            ],
            "OwnerId": "667671192261",
            "GroupId": "sg-2554ce65",
            "IpPermissionsEgress": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": []
                }
            ],
            "VpcId": "vpc-0310bf79"
        }
    ]
}
```

Create new Lambda function

```
$ aws lambda create-function --function-name verifyapi --runtime python3.6
--role arn:aws:iam::667671192261:role/WSU-VERIFY-ROLE --handler app.handler
--code S3Bucket=raspberry-pi-wsu-senior,S3Key=verifyAPI.zip --vpc-config
SubnetIds=subnet-b44713bb,subnet-958b14c9,subnet-ef980a88,subnet-078f1629,s
ubnet-881c3ec2,SecurityGroupIds=sg-2554ce65

Response: {
    "FunctionName": "verifyapi",
    "FunctionArn":
"arn:aws:lambda:us-east-1:667671192261:function:verifyapi",
    "Runtime": "python3.6",
    "Role": "arn:aws:iam::667671192261:role/WSU-VERIFY-ROLE",
    "Handler": "app.handler",
    "CodeSize": 1853710,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-12-06T02:57:39.875+0000",
    "CodeSha256": "zACPXrpoUvAG1wilb9xc7+oH9+IRcoD+xtoc4S+zSR4=",
    "Version": "$LATEST",
    "VpcConfig": {
        "SubnetIds": [
            "subnet-958b14c9",
            "subnet-ef980a88",
            "subnet-078f1629",
            "subnet-881c3ec2",
            "subnet-b44713bb"
        ],
        "SecurityGroupIds": [
            "sg-2554ce65"
        ],
        "VpcId": "vpc-0310bf79"
    },
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "e34e5b2c-d6c6-4d22-8c60-f944c6f15c40"
}
```

Setup API Gateway

Create the API

```
$ aws apigateway create-rest-api --name verifyapi-API
--endpoint-configuration types=REGIONAL

---------------------------------------------------------------------------
Response: {
    "id": "m97wiv8zwh",
    "name": "verifyapi-API",
    "createdDate": 1544065128,
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "REGIONAL"
        ]
    }
}
```

From the previous step, save the “id” parameter and replace it below with API value. Now, you can get the API root resource ID as well.

```
$ API=m97wiv8zwh
$ aws apigateway get-resources --rest-api-id $API

Response: {
    "items": [
        {
            "id": "b0gf9enoh5",
            "path": "/"
        }
    ]
}
```

Copy the id from the response into the clipboard.

Create a Resource in the API

Set the variable ROOT_RESOURCE to the id that you copied from previous step.

```
$ ROOT_RESOURCE=b0gf9enoh5
$ aws apigateway create-resource --rest-api-id $API --path-part
'sensordata' --parent-id $ROOT_RESOURCE

Response: {
    "id": "148fw7",
    "parentId": "b0gf9enoh5",
    "pathPart": "verifyapi",
    "path": "/verifyapi"
}
```

Create Method on the Resource

```
$ RESOURCE=148fw7
$ aws apigateway put-method --rest-api-id $API --resource-id $RESOURCE
--http-method POST --authorization-type "NONE" --api-key-required

Response: {
    "httpMethod": "POST",
    "authorizationType": "NONE",
    "apiKeyRequired": true
}
```

```
$ aws apigateway put-integration --rest-api-id $API --resource-id $RESOURCE
--http-method POST --type AWS_PROXY --integration-http-method POST --uri
arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambd
a:us-east-1:667671192261:function:verifyapi/invocations

Response: {
    "type": "AWS_PROXY",
    "httpMethod": "POST",
    "uri":
"arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lamb
da:us-east-1:667671192261:function:verifyapi/invocations",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "148fw7",
    "cacheKeyParameters": []
}
```

```
$ aws apigateway put-method-response --rest-api-id $API --resource-id $RESOURCE
--http-method POST --status-code 200 --response-parameters
"method.response.header.Access-Control-Allow-Origin=true" --response-models
application/json=Empty

Response: {
    "statusCode": "200",
    "responseParameters": {
        "method.response.header.Access-Control-Allow-Origin": true
    },
    "responseModels": {
        "application/json": "Empty"
    }
}
```

```
$ aws apigateway put-integration-response --rest-api-id $API --resource-id
$RESOURCE --http-method POST --status-code 200 --selection-pattern ""
--response-templates '{"application/json": "{\"json\": \"Empty\"}"}'
--response-parameters
'{"method.response.header.Access-Control-Allow-Origin":"'"'*'"'"}'

Response: {
    "statusCode": "200",
    "selectionPattern": "",
    "responseParameters": {
        "method.response.header.Access-Control-Allow-Origin": "'*'"
    },
    "responseTemplates": {
        "application/json": "{\"json\": \"Empty\"}"
    }
}
```

```
$ aws apigateway create-deployment --rest-api-id $API --stage-name final
--stage-description 'Final Stage' --description 'Deployment for the weather
station pi'

Response: {
    "id": "zhni8q",
    "description": "Deployment for the weather station pi",
    "createdDate": 1543986142
}
```

```
$ aws lambda add-permission --function-name verifyapi --statement-id
apigateway-testt-1 --action lambda:InvokeFunction --principal
apigateway.amazonaws.com --source-arn
"arn:aws:execute-api:us-east-1:667671192261:8h35no6zsg/*/POST/verifyapi"


Response:{
    "Statement":
"{\"Sid\":\"apigateway-testt-1\",\"Effect\":\"Allow\",\"Principal\":{\"Serv
ice\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"
Resource\":\"arn:aws:lambda:us-east-1:667671192261:function:verifyapi\",\"C
ondition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-1:
667671192261:8h35no6zsg/*/POST/verifyapi\"}}}"
}
```

Create API key

```
$ aws apigateway create-api-key --name 'PI-VERIFY-Key' --description 'for
development' --enabled --stage-keys restApiId=$API,stageName=final

Response: {
    "id": "7n2exwj8s5",
    "value": "3wjsITLRVM1iDuPCxCKWc47gWFbTqnqz4JM44NAa",
    "name": "PI-Verify-Key",
    "description": "for development",
    "enabled": true,
    "createdDate": 1543986612,
    "lastUpdatedDate": 1543986612,
    "stageKeys": [
        "m97wiv8zwh/final"
    ]
}
```

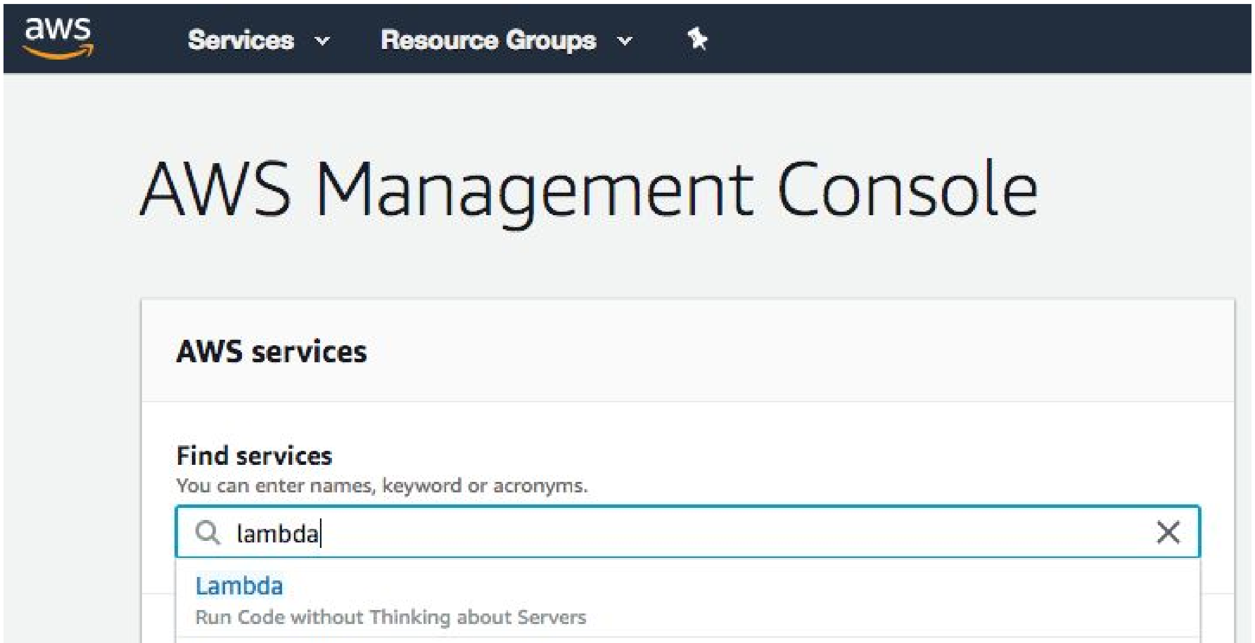
Create usage plan

```
$ aws apigateway create-usage-plan --name "pi-api-key-usagePlan"
--api-stages apiId=$API,stage=final --description "usage plan for pi api
key" --throttle burstLimit=10,rateLimit=150 --quota
limit=500,offset=0,period=MONTH

Response:{
    "id": "mhujyv",
    "name": "pi-api-key-usagePlan",
    "description": "usage plan for pi api key",
    "apiStages": [
        {
            "apiId": "m97wiv8zwh",
            "stage": "final"
        }
    ],
    "throttle": {
        "burstLimit": 10,
        "rateLimit": 150.0
    },
    "quota": {
        "limit": 500,
        "offset": 0,
        "period": "MONTH"
    }
}
```

```
$ USAGEID=mhujyv
$ KEYID=7n2exwj8s5
$ aws apigateway create-usage-plan-key --usage-plan-id $USAGEID --key-type
"API_KEY" --key-id $KEYID

Response: {
    "id": "7n2exwj8s5j",
    "type": "API_KEY",
    "name": "PI-Verify-Key"
}
```

Get your endpoint and x-api-key

Search for lambda



AWS Management Console

AWS services

Find services

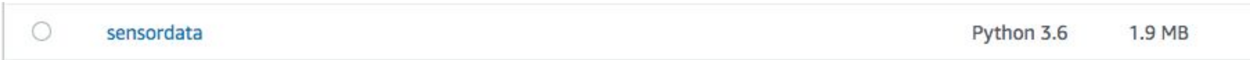
You can enter names, keyword or acronyms.

lambda

Lambda

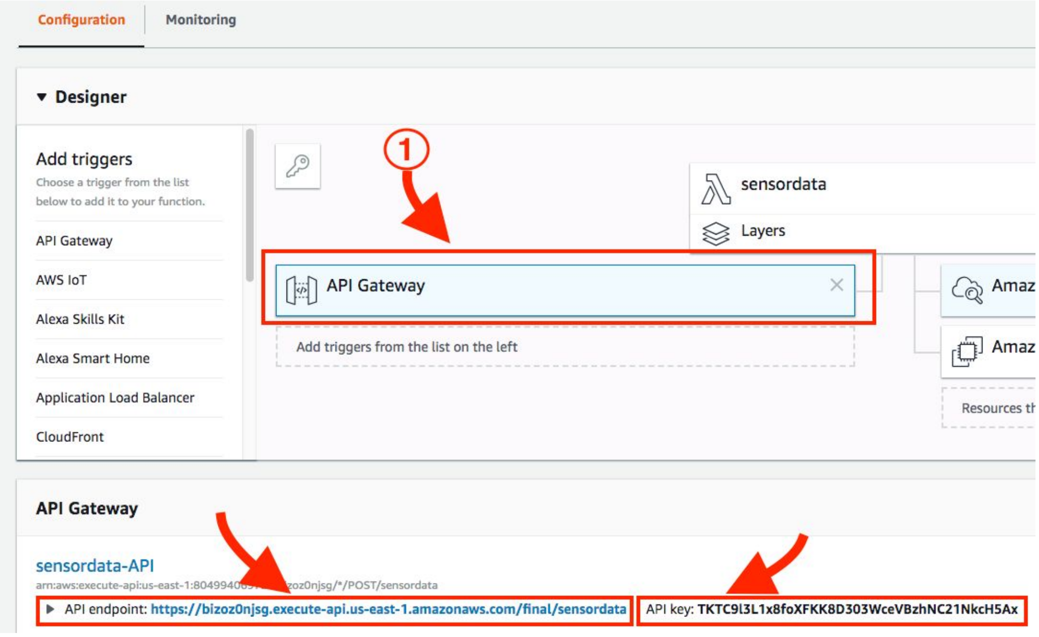
Run Code without Thinking about Servers

Click on sensordata



○	sensordata	Python 3.6	1.9 MB

Click on API Gateway. Make note of the Api Endpoint, and X-API-KEY (in the picture it's API Key)..



API Gateway

sensordata-API

arn:aws:execute-api:us-east-1:80499406 zoz0njsg/*/POST/sensordata

API endpoint: https://bizoz0njsg.execute-api.us-east-1.amazonaws.com/final/sensordata

API key: TKTC9l3L1x8foXFKK8D303WceVBzhNC21NkcH5Ax

Docker

From the “WSU-RaspberryPi-WeatherStation-Final” folder.

```
$ cd deploy
# use nano or vim or vscode or whatever text editor you have in your computer.
I used vim in the below step.
$ echo $ENDPOINT # this command will output your db endpoint. Copy the endpoint
YOUR_DB_ENDPOINT
$ vim knexfile.js
```

Change the “host” value to the address you copied from previous step. And save the file.

```
host : 'wsu-station-rds.cxzjmoez6mye.us-east-1.rds.amazonaws.com'
```

Build

From the directory “deploy” execute the following line. When it completes building it. Run the same line again.

```
$ docker build -t final .
```

```
$ docker images
```

Push to the hub

Find the image name that says “final” and copy the IMAGE ID of it. Assign it to the IMAGEID.

```
$ IMAGEID=0fbf759e9caf
$ docker tag $IMAGEID yourhubusername/final:latest
$ docker push yourhubusername/final:latest
```

AWS Deployment

Add docker-hub to prerequisite and docker installations on local machine
Write a simple shell file to build docker image
Launch instance instructions

```
$ aws --profile $PROFILE ec2 create-security-group --group-name
AutoScaling-SG --description "ASG security group"

Response: {
    "GroupId": "sg-0102b5d81b301813c"
}
```

Copy the GroupId from the above response.

```
$ GROUPID=sg-0102b5d81b301813c
```

Create key pair

```
$ aws --profile $PROFILE ec2 create-key-pair --key-name US-KeyPair --query
'KeyMaterial' --output text > US-KeyPair.pem
```

```
$ chmod 600 US-KeyPair.pem
```

Run an Instance

```
$ aws --profile $PROFILE ec2 run-instances --image-id ami-0ac019f4fcb7cb7e6
--count 1 --instance-type t2.micro --key-name US-KeyPair
--security-group-ids $GROUPID

Response: {
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0ac019f4fcb7cb7e6",
            "InstanceId": "i-038b40a97e9a31339",
            "InstanceType": "t2.micro",
            "KeyName": "US-KeyPair",
            "LaunchTime": "2018-12-07T04:28:53.000Z",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-1c",
                "GroupName": "",
                "Tenancy": "default"
            },
        }...
```

Copy the instance id from the response above

```
$ INSTANCE=i-038b40a97e9a31339
$ aws --profile $PROFILE ec2 describe-instance-status --instance-id
$INSTANCE

Response: {
    "InstanceStatuses": [
        {
            "AvailabilityZone": "us-east-1a",
            "InstanceId": "i-038b40a97e9a31339",
            "InstanceState": {
                "Code": 16,
                "Name": "running"
            },...
}
```

Pay attention to the InstanceState. If the "Code" is 16, move on to the next step, otherwise wait and retry this accept until you see the "Code" 16.

Update Security Group

```
$ aws --profile $PROFILE ec2 describe-instances --instance-ids $INSTANCE
--query 'Reservations[*].Instances[*].PublicIpAddress' --output text
```

copy the text to the clipboard

```
$ IPV4=54.204.181.162
```

```
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-name
AutoScaling-SG --protocol tcp --port 22  --cidr 0.0.0.0/0
```

```
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-name
AutoScaling-SG --protocol tcp --port 80  --cidr 0.0.0.0/0
```

```
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-name
AutoScaling-SG --protocol tcp --port 443  --cidr 0.0.0.0/0

$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-name
AutoScaling-SG --protocol tcp --port 8000  --cidr 0.0.0.0/0

$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-name
AutoScaling-SG --protocol tcp --port 5000  --cidr 0.0.0.0/0
```

```
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-id
$VPCSG --protocol tcp --port 3306  --source-group AutoScaling-SG
```

Add the Instance Private IP address in Security Group

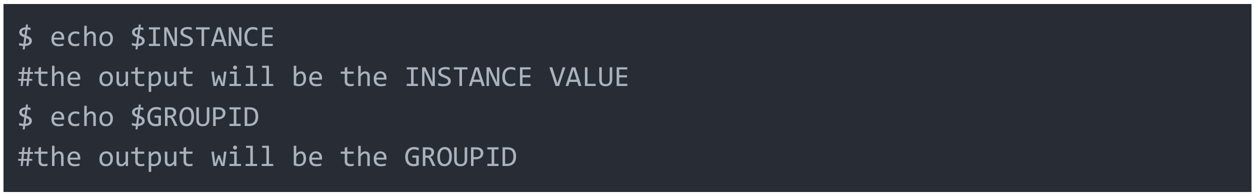
Retrieve the private IP address

```
$ aws --profile $PROFILE ec2 describe-instances --instance-ids $INSTANCE
--query 'Reservations[*].Instances[*].PrivateIpAddress' --output text
```

Add the private ip address of the ec2 into the RDS Security group

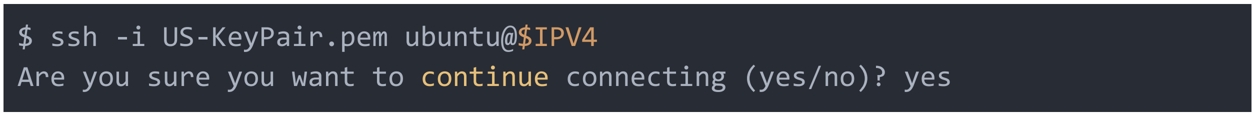
```
$ PRIVATE=172.31.32.83/32
$ aws --profile $PROFILE ec2 authorize-security-group-ingress --group-id
$VPCSG --protocol tcp --port 3306  --cidr $PRIVATE
```

Run the following command. Copy the output text and save it in safe place. I will refer this value as INSTANCE VALUE and GROUPID next time when you'd require it again.

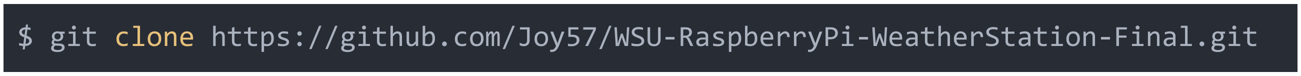


```
$ echo $INSTANCE
#the output will be the INSTANCE VALUE
$ echo $GROUPID
#the output will be the GROUPID
```

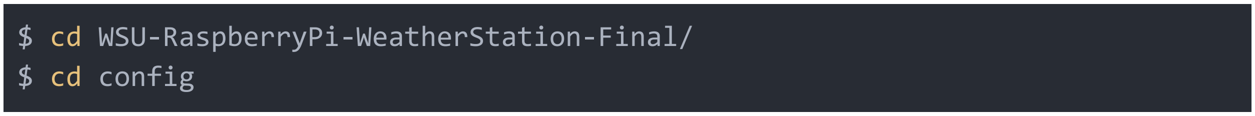
SSH



```
$ ssh -i US-KeyPair.pem ubuntu@$IPV4
Are you sure you want to continue connecting (yes/no)? yes
```

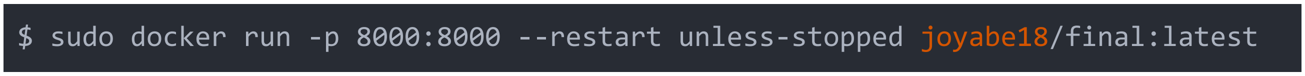


```
$ git clone https://github.com/Joy57/WSU-RaspberryPi-WeatherStation-Final.git
```

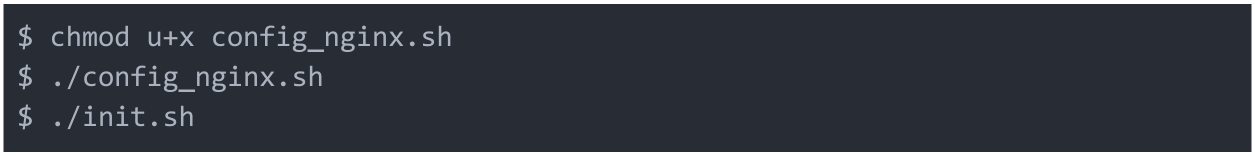


```
$ cd WSU-RaspberryPi-WeatherStation-Final/
$ cd config
```

In the config folder, open init.sh file and change the line where it says "joyabe18" to your docker user name.

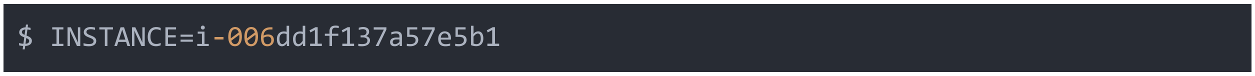


```
$ sudo docker run -p 8000:8000 --restart unless-stopped joyabe18/final:latest
```



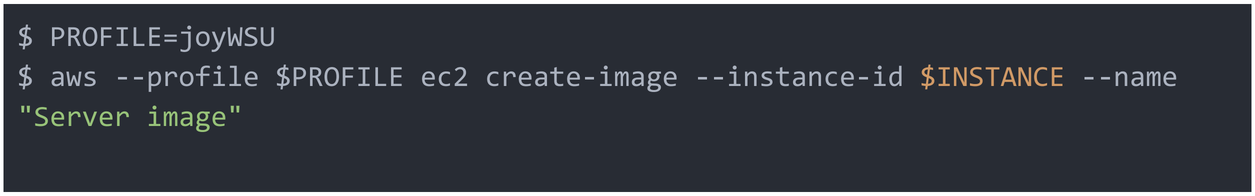
```
$ chmod u+x config_nginx.sh
$ ./config_nginx.sh
$ ./init.sh
```

open a new terminal window and execute the following:
Now use the INSTANCE VALUE you copied in the previous steps. And assign it to INSTANCE.



```
$ INSTANCE=i-006dd1f137a57e5b1
```

Set the PROFILE value to joyWSU



```
$ PROFILE=joyWSU
$ aws --profile $PROFILE ec2 create-image --instance-id $INSTANCE --name
"Server image"
```

```
Response: {
    "ImageId": "ami-08536a237f1396438"
}
```

copy the above ImageId and assign it to IMAGEID. You'll use it again later.

```
$ IMAGEID=ami-08536a237f1396438
```

Elastic Load Balancer

Make sure you're in the "WSU-RaspberryPi-WeatherStation-Final"

Create Policy

```
$ aws --profile $PROFILE iam create-policy --policy-name
loadbalancer-policy --policy-document file://policies/ALB.json

Response: {
    "Policy": {
        "PolicyName": "loadbalancer-policy",
        "PolicyId": "ANPAIGMEOXOSIGEIEK3PC",
        "Arn": "arn:aws:iam::667671192261:policy/loadbalancer-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2018-12-06T19:48:09Z",
        "UpdateDate": "2018-12-06T19:48:09Z"
    }
}
```

Create role

```
$ aws --profile $PROFILE iam create-role --role-name WSU-ELB-ROLE
--assume-role-policy-document file://policies/trusty.json

Response: {
    "Role": {
        "Path": "/",
        "RoleName": "WSU-ELB-ROLE",
        "RoleId": "AROAJCFAHSI2GZ5EFRPZG",
        "Arn": "arn:aws:iam::667671192261:role/WSU-ELB-ROLE",
        "CreateDate": "2018-12-06T19:49:21Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Attach policy

```
$ aws --profile $PROFILE iam attach-role-policy --role-name WSU-ELB-ROLE
--policy-arn arn:aws:iam::YOUR_ACCOUNT_ID:policy/loadbalancer-policy
```

Update the script elb.py

```
$ echo $IMAGEID
```

Copy the outputted text.
Make sure you're in the "WSU-RaspberryPi-WeatherStation-Final".

```
$ cd lambda
```

Open the elb.py script in a text editor and change the **ImageId** value.

```
def create_launch_config(security_group_ID, key_Name):
   response = autoscaling.create_launch_configuration(
   ImageId='ami-05baba7b0f1bc95ce',
   KeyName= key_Name,
   InstanceType='t2.micro',
   LaunchConfigurationName='lambda-launch-config-test1',
   SecurityGroups=[
       security_group_ID,
   ],
)
```

Replace the “security_group_ID” value with the GROUPID value below that you copied before.

```
def lambda_handler(event, context):
    id = getVpcIds()
    print("vpc_id--->", id)
    subnets = getSubnets(id)
    print(subnets)
    security_group_ID = "sg-0a0876f20bed41e27"
```

Now zip the file and name it loadbalancer.zip. And Make sure you’re in the “lambda” directory.

```
$ aws --profile $PROFILE s3 cp loadbalancer.zip
s3://raspberry-pi-wsu-senior
```

Create Lambda function

Remember to replace the YOUR_ACCOUNT_ID

```
$ aws --profile $PROFILE lambda create-function --function-name ELB
--runtime python3.6 --role arn:aws:iam::YOUR_ACCOUNT_ID:role/WSU-ELB-ROLE
--handler elb.lambda_handler --code
S3Bucket=raspberry-pi-wsu-senior,S3Key=loadbalancer.zip

Response: {
     "FunctionName": "ELB",
     "FunctionArn": "arn:aws:lambda:us-east-1:667671192261:function:ELB",
     "Runtime": "python3.6",
     "Role": "arn:aws:iam::667671192261:role/WSU-ELB-ROLE",
     "Handler": "lambda_function.lambda_handler",
     "CodeSize": 1333,
     "Description": "",
     "Timeout": 3,
     "MemorySize": 128,
     "LastModified": "2018-12-06T19:51:46.328+0000",
     "CodeSha256": "0j4VeVR4aNAx0YRRQ2SWDnyQJnR1/8y8vrLGWzGhUjY=",
     "Version": "$LATEST",
     "TracingConfig": {
         "Mode": "PassThrough"
     },
     "RevisionId": "504ae586-2e9d-4c78-a89c-a0a6e2ac74d6"
}
```

Auto Scaling Group

Make sure you're in the "WSU-RaspberryPi-WeatherStation-Final" directory.

Create IAM policy

```
$ aws --profile $PROFILE iam create-policy --policy-name autoscaling-policy
--policy-document file://policies/ASG.json

Response: {
    "Policy": {
        "PolicyName": "autoscaling-policy",
        "PolicyId": "ANPAIP7B34DTPXKTXBYPW",
        "Arn": "arn:aws:iam::667671192261:policy/autoscaling-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2018-12-06T20:16:01Z",
        "UpdateDate": "2018-12-06T20:16:01Z"
    }
}
```

Create IAM role

```
$ aws --profile $PROFILE iam create-role --role-name WSU-ASG-ROLE
--assume-role-policy-document file://policies/trusty.json

Response: {
    "Role": {
        "Path": "/",
        "RoleName": "WSU-ASG-ROLE",
        "RoleId": "AROAIIG3UEBSRLEJV6IKA",
        "Arn": "arn:aws:iam::667671192261:role/WSU-ASG-ROLE",
        "CreateDate": "2018-12-06T20:16:56Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Attach Policy

```
$ aws --profile $PROFILE iam attach-role-policy --role-name WSU-ASG-ROLE
--policy-arn arn:aws:iam::YOUR_ACCOUNT_ID:policy/autoscaling-policy
```

Make sure you're in the "WSU-RaspberryPi-WeatherStation-Final" directory.

```
cd lambda
```

Copy the "TargetGroupArn" from the response below.

```
$ aws --profile $PROFILE elbv2 describe-target-groups
{
    "TargetGroups": [
        {
            "TargetGroupArn":
"arn:aws:elasticloadbalancing:us-east-1:804994069721:targetgroup/my-targets
-test2/cb3c2e82b2a7dc83",
            "TargetGroupName": "my-targets-test2",
            "Protocol": "HTTP",
            "Port": 80,
            "VpcId": "vpc-ca8ea2b1",
            "HealthCheckProtocol": "HTTP",
            "HealthCheckPort": "traffic-port",
            "HealthCheckEnabled": true,
            "HealthCheckIntervalSeconds": 30,
            "HealthCheckTimeoutSeconds": 5,
            "HealthyThresholdCount": 5,
            "UnhealthyThresholdCount": 2,
            "HealthCheckPath": "/",
            "Matcher": {
                "HttpCode": "200"
            },
            "LoadBalancerArns": [

"arn:aws:elasticloadbalancing:us-east-1:804994069721:loadbalancer/app/loadb
alancer-joy-final/ce2ef6d71edf3790"
            ],
            "TargetType": "instance"
        }
    ]
}
```

Open the file called asg.py and replace the TargetGroupARNs value with the one you copied earlier.

```
    HealthCheckType='ELB',
    LaunchConfigurationName='lambda-launch-config-test1',
    TargetGroupARNs=[
        'NEW_TARGET_GROUP_ARN',
    ],
    MaxSize=2,
    MinSize=2,
)
```

ZIP the asg.py file and name it autoscaling.zip.
From the directory “lambda”, execute the following command.

```
$ aws --profile $PROFILE s3 cp autoscaling.zip s3://raspberry-pi-wsu-senior
```

Create Lambda function

```
$ aws --profile $PROFILE lambda create-function --function-name ASG
--runtime python3.6 --role arn:aws:iam::YOUR_ACCOUNT_ID:role/WSU-ASG-ROLE
--handler asg.lambda_handler --code
S3Bucket=raspberry-pi-wsu-senior,S3Key=autoscaling.zip

Response: {
    "FunctionName": "ASG",
    "FunctionArn": "arn:aws:lambda:us-east-1:667671192261:function:ASG",
    "Runtime": "python3.6",
    "Role": "arn:aws:iam::667671192261:role/WSU-ASG-ROLE",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 553,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2018-12-06T20:18:41.439+0000",
    "CodeSha256": "ugKM/m3G94OU3bq7a4IgyNko3sQIAW+8OED2CSV4SS8=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "f247c535-96d3-4a00-a2c2-5c25c13fa27d"
}
```

INVOKE ENDPOINTS

Deploy load balancer

```
$ aws --profile $PROFILE lambda invoke --function-name
arn:aws:lambda:us-east-1:YOUR_ACCOUNT_ID:function:ELB outfile.txt
```

Deploy AutoScaling group

```
$ aws --profile $PROFILE lambda invoke --function-name
arn:aws:lambda:us-east-1:YOUR_ACCOUNT_ID:function:ASG outfile.txt
```