# AWS MANUAL

Version 1.0 Final

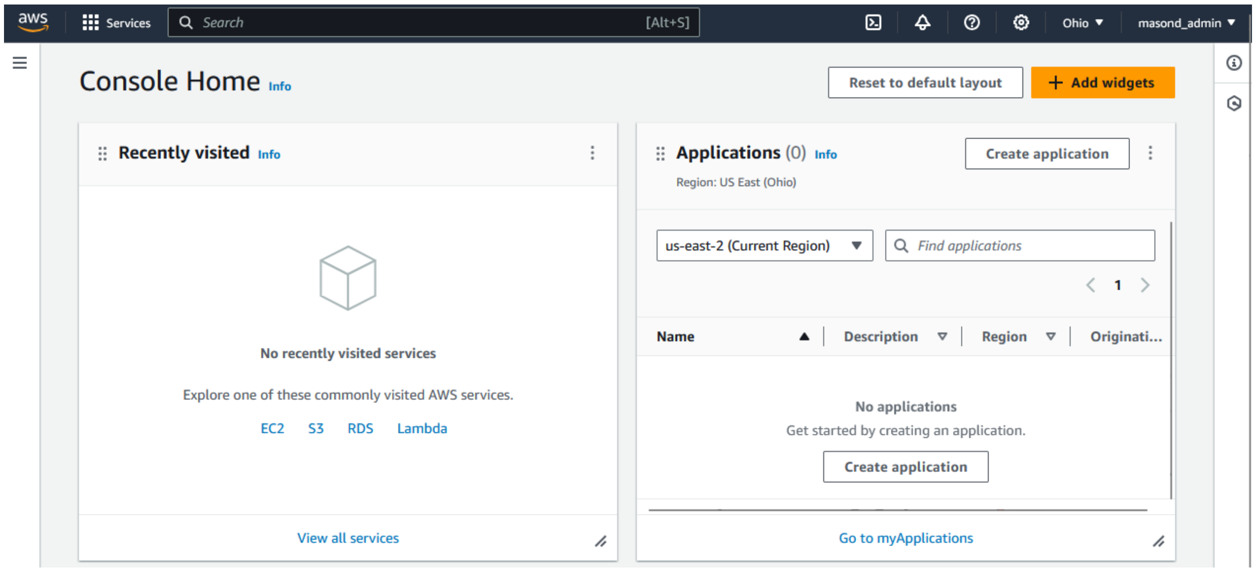
Mason Davis
Masondavis4@suu.edu
December 6th, 2024

**Table of Contents**

## Contents

# Chapter 1: AWS API/CLI

## Exercise 1.1: Create an AWS Account.



No recently visited services

Explore one of these commonly visited AWS services.

No applications

Get started by creating an application.

## **Exercise 1.2:** Create an IAM Administrators Group and User

In this exercise, you'll define an Administrators group and then add a user to that group. Generate API keys for this user and call this user DevAdmin.

1. Sign in to the AWS Management Console (at signin.aws.amazon.com/console).
2. Select **All Services**.
3. To open the IAM dashboard, select **IAM**.
4. To view the list of IAM groups, select **Groups**.

If this is a new account, the list is empty.

5. Choose **Create New Group**.
6. For **Group Name**, enter **Administrators**.
7. Choose **Next Step**.
8. On the **Attach Policy** page, select the AdministratorAccess policy.
9. Choose **Next Step**.
10. On the **Review** page, choose **Create Group** to create the Administrators group.
11. To view the list of IAM users, select **Users**.

If this is a new account, the list is empty.

12. Choose **Add user**.
13. Set the user name to **DevAdmin**.

14. Select both Access type check boxes: **Programmatic access** and **AWS Management Console access**.
15. Choose **Next: Permissions**.
16. To add this user to the Administrators group, select the **Administrators group** check box.
17. Clear the **Require password reset** check box.
18. Choose **Next: Tags**.
19. Provide a tag with a key of **project** and a value of **dev-study-guide**.

Use tags to add customizable key-value pairs to resources so that you can more easily track and manage them.

20. Choose **Next: Review**.
21. Choose **Create user**.
22. Download the credentials.csv file.
23. Rename the file to **devadmin-credentials.csv**, and move the file to a folder where you would like to keep it.
24. Sign out of the AWS Management Console by clicking your name in the top bar and selecting **Sign Out**.

You now have a .csv file that contains a user name, password, access key ID, secret access key, and console login link. Use the DevAdmin user name, password, and console sign-in link to sign in to the AWS Management Console for all future exercises unless otherwise noted. Use the access key to configure the SDK in the following exercises.

## **Exercise 1.3:** Install and Configure the AWS CLI

In this exercise, you'll install and configure the AWS Command Line Interface (AWS CLI). The AWS CLI requires Python2 or Python3. Install Python using pip, the Python installer.

1. Install Python from https://www.python.org/downloads/.
2. Open a terminal window.
3. To install the AWS CLI, run the following command:

```
pip install aws-cli --upgrade --user
```

4. (Optional) If you encounter issues with step 3, review the AWS CLI Installation guide for alternative installation options here:

https://docs.aws.amazon.com/cli/latest/userguide/installing.html

5. To configure the AWS CLI with a default profile for credentials, run the following command:

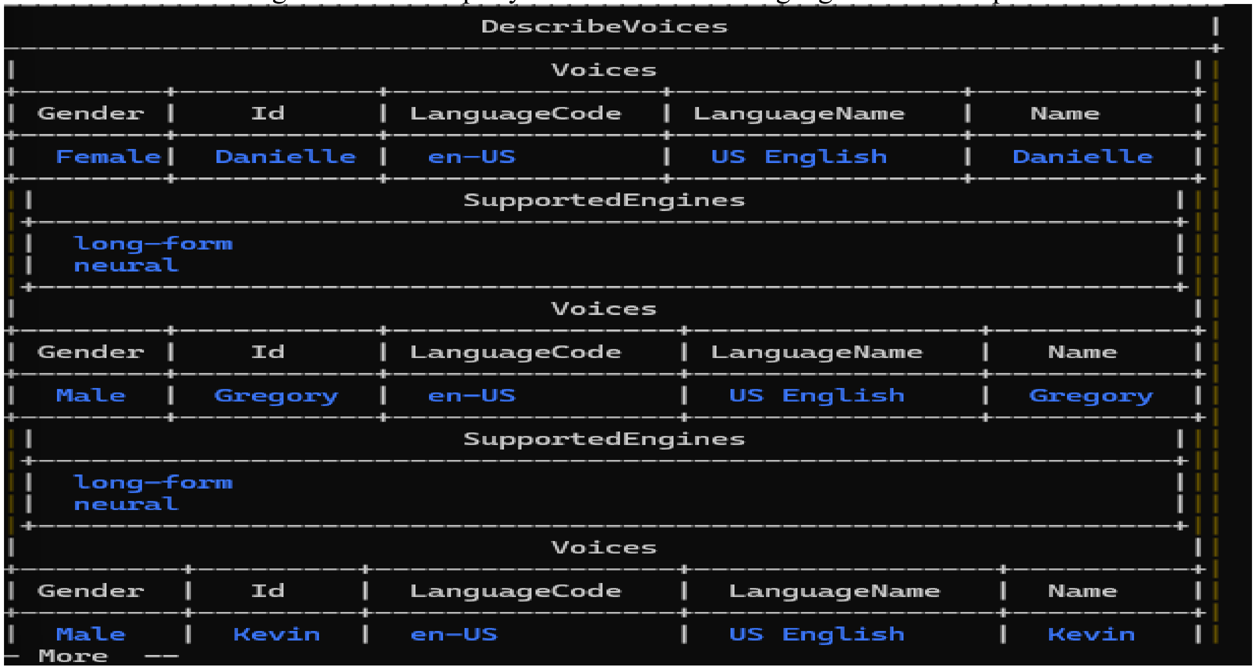
```
aws configure
```

6. Enter the following values when prompted:

1. AWS Access Key ID: Paste the value from the CSV you downloaded in Exercise 1.2.
2. AWS Secret Access Key: Paste the value from the CSV you downloaded in Exercise 1.2.

3. Default region name: Enter us-east-1.

4. Default output format: Press Enter to leave this blank. Set up access key and secret to set up CLI and confirm its working with the "aws polly describe-voices --language en-US --output table" Command



```
|                              DescribeVoices                              |
+--------------------------------------------------------------------------+
|                                  Voices                                  |
+----------+------------+---------------+-----------------+----------------+
| Gender   |     Id     | LanguageCode  | LanguageName    |      Name      |
+----------+------------+---------------+-----------------+----------------+
|  Female|  Danielle  |   en-US       |  US English     |  Danielle      |
+----------+------------+---------------+-----------------+----------------+
||                            SupportedEngines                            ||
|+------------------------------------------------------------------------+|
||   long-form                                                            ||
||   neural                                                               ||
|+------------------------------------------------------------------------+|
|                                  Voices                                  |
+----------+------------+-----------------+-----------------+--------------+
| Gender   |     Id     | LanguageCode    |  LanguageName   |     Name     |
+----------+------------+-----------------+-----------------+--------------+
|  Male    |  Gregory   |   en-US         |   US English    |  Gregory     |
+----------+------------+-----------------+-----------------+--------------+
||                            SupportedEngines                            ||
|+------------------------------------------------------------------------+|
||   long-form                                                            ||
||   neural                                                               ||
|+------------------------------------------------------------------------+|
|                                  Voices                                  |
+-----------+---------+------------------+------------------+--------------+
| Gender    |   Id    | LanguageCode     |   LanguageName   |  Name        |
+-----------+---------+------------------+------------------+--------------+
|  Male     |  Kevin  | en-US            |   US English     |  Kevin       |
- More  --
```

## Exercise 1.4: Downloaded code snippets

1. If you haven't downloaded the Chapter Resources from the online test bank, go to https://www.wiley.com/go/sybextestprep.
2. Register to get an access code.
3. Login and then redeem the access code. The book will be added to the online test bank.
4. Next to "Course Dashboard," click Resources.
5. Click "Chapter Resources" to download the code files.

## Exercise 1.5: Run a Python Script that Makes AWS API Calls

In this exercise, you'll run the Python script to make an AWS API call.

1. Open a terminal window and navigate to the folder with the book sample code.
2. To install the AWS SDK for Python (Boto), run the following command:

pip install boto3

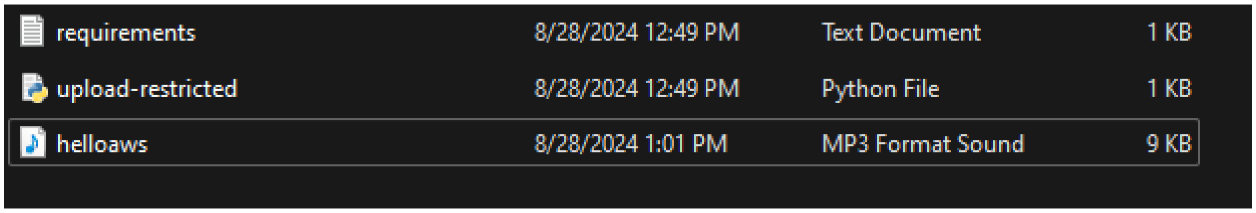
3. Navigate to the chapter-01 folder where you downloaded the sample code.
4. To generate an MP3 in the chapter-01 folder, run the helloworld.py program.

python helloworld.py

5. To hear the audio, open the generated file, helloworld.mp3.

6. (Optional) Modify the Python code to use a different voice. See Exercise 1.3 for an AWS CLI command that provides the list of available voices.

You hear "Hello World" when you play the generated audio file. If you completed the optional challenge, you also hear the audio spoken in a different voice from the first audio.



```
requirements       8/28/2024 12:49 PM   Text Document      1 KB
upload-restricted  8/28/2024 12:49 PM   Python File        1 KB
helloaws           8/28/2024 1:01 PM    MP3 Format Sound   9 KB
```

## **Exercise 1.6**: Working with Multiple Regions

In this exercise, you'll use Amazon Polly to understand the effects of working with different AWS Regions.

1. Open a terminal window and navigate to the folder with the book sample code.
2. Navigate to chapter-01 in the folder where you downloaded the sample code.
3. Verify that the region is us-east-1 by running the following command:

aws configure get region

4. Upload aws-lexicon.xml to the Amazon Polly service in the default region, which is US East (N. Virginia).

aws polly put-lexicon --name awsLexicon --content file://aws-lexicon.xml

5. The file helloaws.py is currently overriding the region to be EU (London). Run the Python code and observe the LexiconNotFoundException that returns.

python.helloaws.py

6. Upload the lexicon to EU (London) by setting the region to eu-west-2.

aws polly put-lexicon --name awsLexicon --content  file://aws-lexicon.xml --region eu-west-2

7. Run the following Python script again:

python helloaws.py

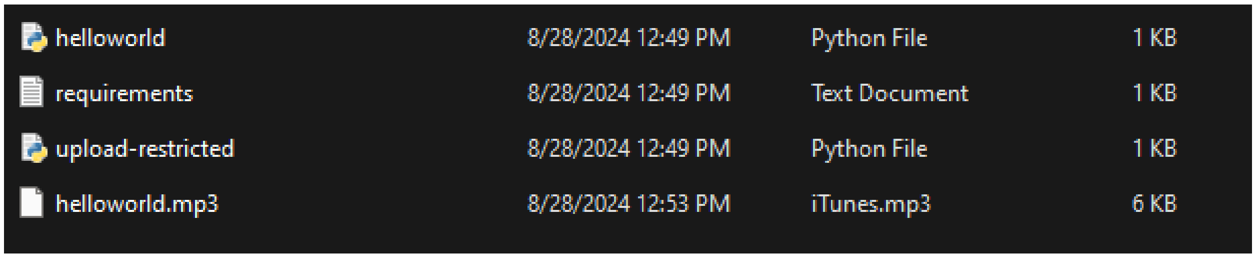
Observe that it executes successfully this time and generates an MP3 file in the current folder.

8. Play the generated helloaws.mp3 file to confirm that it says, "Hello Amazon Web Services."
9. (Optional) Delete the lexicons with the following commands:
10. aws polly delete-lexicon --name awsLexicon

aws polly delete-lexicon --name awsLexicon --region eu-west-2

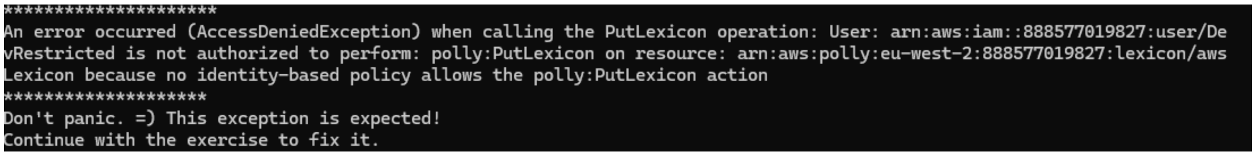
Even though the text supplied by the API call to synthesize speech was "Hello AWS!," the generated audio file uses the lexicon you uploaded to pronounce it as "Hello Amazon Web Services."

created us east 1 and eu west 2 region, observed error when aws was called without being put in region, then it worked when it was assigned to the lexicicon in eu-west-2. Voice recording says hello a w s.



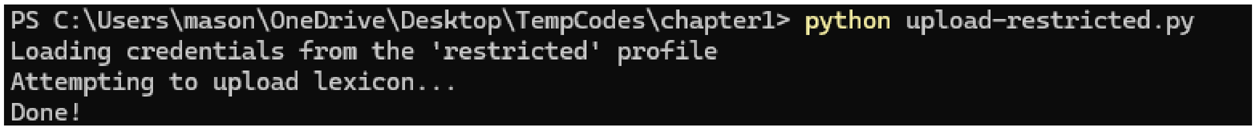
```
helloworld          8/28/2024 12:49 PM   Python File     1 KB
requirements        8/28/2024 12:49 PM   Text Document   1 KB
upload-restricted   8/28/2024 12:49 PM   Python File     1 KB
helloworld.mp3      8/28/2024 12:53 PM   iTunes.mp3      6 KB
```

**Exercise 1.7:** Set up a new user with restricted access. Ran python upload-restricted.py



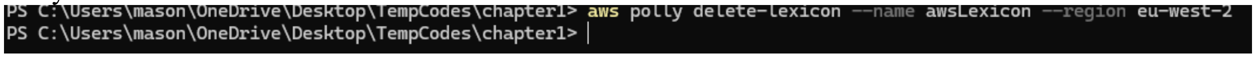
```
*********************
An error occurred (AccessDeniedException) when calling the PutLexicon operation: User: arn:aws:iam::888577019827:user/De
vRestricted is not authorized to perform: polly:PutLexicon on resource: arn:aws:polly:eu-west-2:888577019827:lexicon/aws
Lexicon because no identity-based policy allows the polly:PutLexicon action
********************
Don't panic. =) This exception is expected!
Continue with the exercise to fix it.
```

next changedp permission to full, than ran the same code.



```
PS C:\Users\mason\OneDrive\Desktop\TempCodes\chapter1> python upload-restricted.py
Loading credentials from the 'restricted' profile
Attempting to upload lexicon...
Done!
```

Fully removed lexicon afterwards



```
PS C:\Users\mason\OneDrive\Desktop\TempCodes\chapter1> aws polly delete-lexicon --name awsLexicon --region eu-west-2
PS C:\Users\mason\OneDrive\Desktop\TempCodes\chapter1> |
```

Always remember to clean up after yourself following asses/object creation.

# Chapter 2: Introduction to Computing and Networking

## **Exercise 2.1:** Create an Amazon EC2 Key Pair

In this exercise, you'll generate and save an Amazon EC2 key pair. You are responsible for saving the private key and using it when you want to connect to your Amazon EC2 instances.

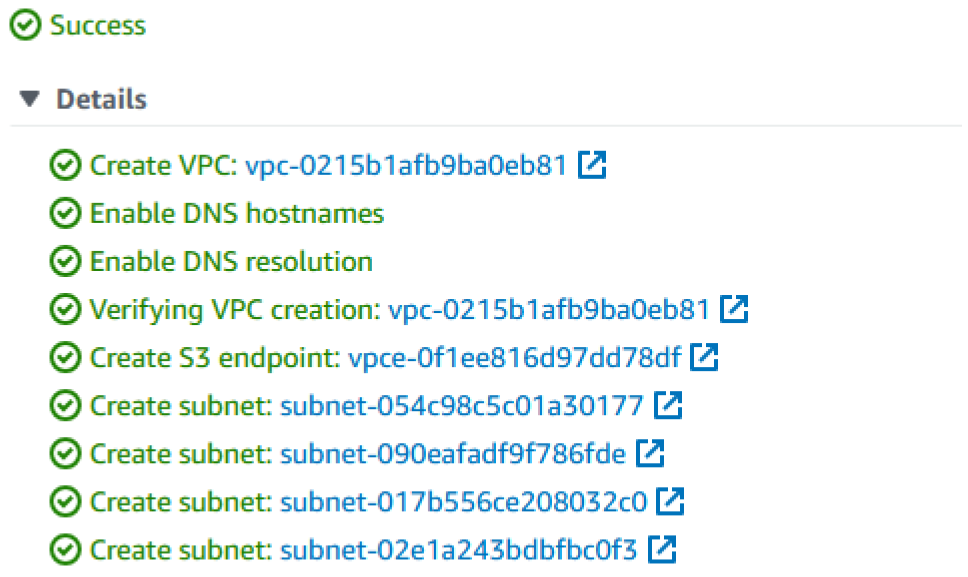
Under services, EC2, Network and Security, Key Pairs, Create  keypair.

devassoc.ppk 9/8/2024 10:48 AM

## **Exercise 2.2:** Create an Amazon VPC with Public and Private Subnets

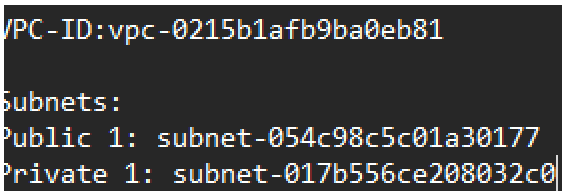
In this exercise, you'll create an Amazon Virtual Private Cloud (Amazon VPC). Within that Amazon VPC, you will have a public subnet directly connected to the internet through an internet gateway. You will also have a private subnet that only has an indirect connection to the internet using network address translation (NAT).

Created the VPC called devassoc with an AWS NAT through Arizona.


Success

Details

- Create VPC: vpc-0215b1afb9ba0eb81
- Enable DNS hostnames
- Enable DNS resolution
- Verifying VPC creation: vpc-0215b1afb9ba0eb81
- Create S3 endpoint: vpce-0f1ee816d97dd78df
- Create subnet: subnet-054c98c5c01a30177
- Create subnet: subnet-090eafadf9f786fde
- Create subnet: subnet-017b556ce208032c0
- Create subnet: subnet-02e1a243bdbfbc0f3


```
PC-ID:vpc-0215b1afb9ba0eb81

ubnets:
ublic 1: subnet-054c98c5c01a30177
rivate 1: subnet-017b556ce208032c0
```

## Exercise 2.3: Use an IAM Role for API Calls from Amazon EC2 Instances

In this exercise, you'll create an IAM role for the web server. This role enables you to make AWS Cloud API calls from code running on the Amazon EC2 instance of the web server. You are not required to save IAM credentials in a file on the instance. To do this, create a new IAM role and call it the devassoc-webserver role. The role provides permissions needed for the API calls.

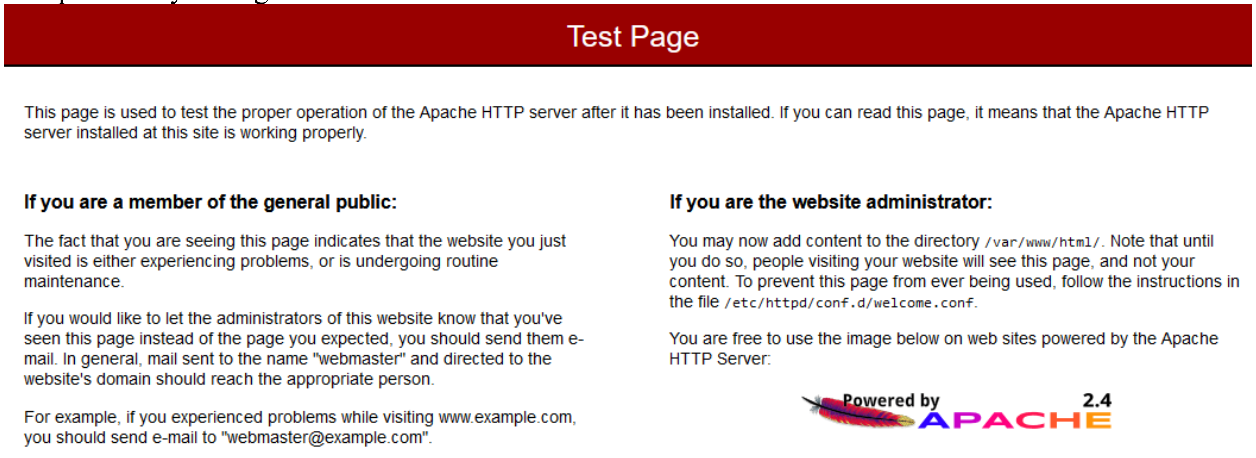
1. Select **Services ➢ IAM**.
2. Select **Roles** and choose **Create Role**.
3. Under **Choose the service that will use this role**, select the option that allows Amazon EC2 instances to call AWS services on your behalf, and then choose **Next: Permissions**.
4. Select the following AWS managed policies to attach them to the devassoc- webserver role and then choose **Next: Tags**:

- AmazonPollyReadOnlyAccess: Grant read-only access to resources, list lexicons, fetch lexicons, list available voices, and synthesize speech to apply lexicons to the synthesized speech.
- TranslateReadOnly: Allow permissions to detect the dominant language in text, translate text, and list and retrieve custom terminologies.

Created devassoc-webserver role in IAM menu.

## Exercise 2.4: Set up EC2

you should be able to connect after attaching resources, and making sure access points are availabe and set up correctly through the VPC



Test Page

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.

**If you are a member of the general public:**

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

**If you are the website administrator:**

You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the image below on web sites powered by the Apache HTTP Server:

## Exercise 2.5: Connect to the Amazon EC2 Instance

In this exercise, you'll connect to the Amazon EC2 Instance using SSH.

1. Select **Services** > **EC2**.
2. Select **Instances.**

3. Select the **webserver** instance from the list of instances.
4. Select **Actions** > **Connect**.
5. In the **Connect to Your Instance** dialog box, follow the directions to establish an SSH connection.
6. From within your SSH session, run this command to view the available metadata fields from the Amazon EC2 metadata service:

```
curl 169.254.169.254/latest/meta-data/
```

7. Run this command to query the Amazon EC2 instance ID:

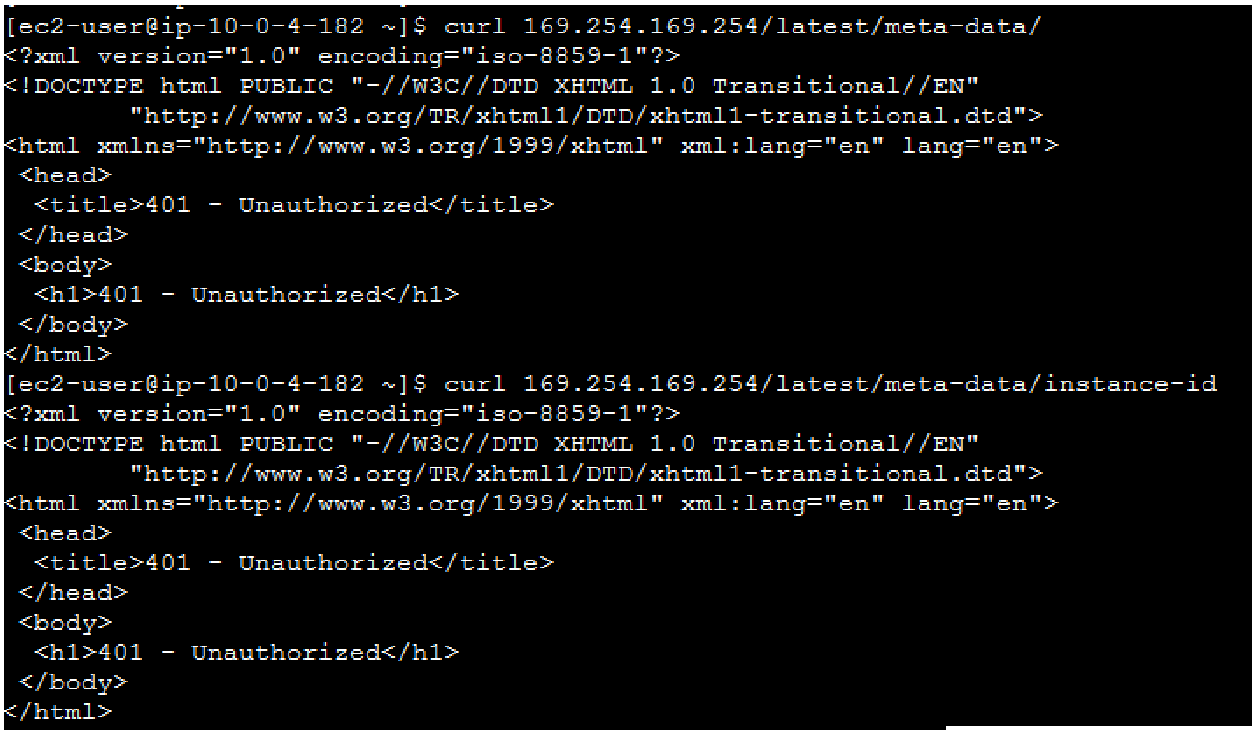
```
curl 169.254.169.254/latest/meta-data/instance-id
```

8. Call **AWS Cloud API** using the AWS CLI. This command translates text from English to French and uses credentials from the AWS role you assigned to the instance. Enter the following command as a single line:

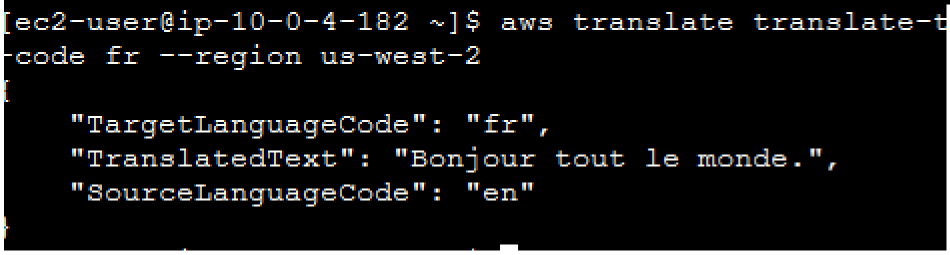
```
aws translate translate-text --text "Hello world." --source-language-code  en --target-language-code fr --
region us-west-2
```

9. To review the credentials that are being passed to the instance, query the Amazon EC2 Metadata service:

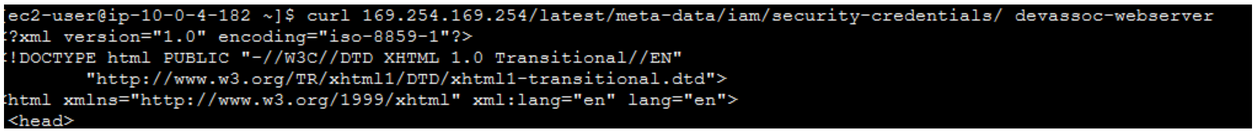
```
curl 169.254.169.254/latest/meta-data/iam/security-credentials/ deva
```



```
[ec2-user@ip-10-0-4-182 ~]$ curl 169.254.169.254/latest/meta-data/
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title>401 - Unauthorized</title>
 </head>
 <body>
  <h1>401 - Unauthorized</h1>
 </body>
</html>
[ec2-user@ip-10-0-4-182 ~]$ curl 169.254.169.254/latest/meta-data/instance-id
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title>401 - Unauthorized</title>
 </head>
 <body>
  <h1>401 - Unauthorized</h1>
 </body>
</html>
```



```
[ec2-user@ip-10-0-4-182 ~]$ aws translate translate-t
-code fr --region us-west-2
{
    "TargetLanguageCode": "fr",
    "TranslatedText": "Bonjour tout le monde.",
    "SourceLanguageCode": "en"
}
```



```
ec2-user@ip-10-0-4-182 ~]$ curl 169.254.169.254/latest/meta-data/iam/security-credentials/ devassoc-webserver
?xml version="1.0" encoding="iso-8859-1"?>
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
```

## **Exercise 2.6:** Configure NAT for Instances in the Private Subnet

In this exercise, you'll create a security group for the NAT instance. NAT allows Amazon EC2 instances in the private subnet to make web requests to the internet, to update software packages, and to make API calls.

1. Select **Services** > **VPC**.
2. From the **Security** section, select **Security Groups**.
3. Select **Create Security Group** and configure the properties as follows:

1. Set the **Name** tag to nat-sg.
2. Set the **Group name** to nat-sg.
3. Set the **Description** to Allow NAT instance to forward internet traffic.

4. Set **Amazon VPC** to devassoc.

4. Choose **Create** to save the group, and then choose **Close** to return to the list of security groups.
5. Select the **nat-sg** security group.
6. To modify the inbound rules, select the **Inbound Rules** tab and select **Edit rules**.
7. Select **Add Rule**, and set the following properties for the first rule:

1. From **Type**, select **HTTP (80)**.
2. For **Source**, enter **10.0.0.0/16**.
3. For **Description**, enter **Enable internet bound HTTP requests from VPC instances**.

8. Select **Add Rule**, and set the following properties for this rule:

1. From **Type**, select **HTTPS (443)**.
2. For **Source**, enter **10.0.0.0/16**.
3. For **Description**, enter **Enable internet bound HTTPS**. **requests from VPC instances**.

9. Select **Add Rule**, and set the following properties for this rule:

1. From **Type**, select **All ICMP – IPv4**.
2. For **Source**, enter **10.0.0.0/16**.
3. For **Description**, **Enable outbound PING requests from VPC instances**.

## **Exercise 2.7:** Launch an Amazon EC2 Instance into the Private Subnet

In this exercise, you'll launch an Amazon EC2 instance into the private subnet and then verify that the security group allows HTTP from anywhere. Because this instance is in the private subnet, it does not have a public IP address. Even though the instance can make outbound requests to the internet through the NAT instance, it is not reachable for inbound connections from the internet.

1. Select **Services** > **EC2**.
2. Choose **Launch Instance**.
3. Select **Amazon Linux 2 AMI**.
4. Select **t2.micro** and then choose **Next: Configure Instance Details**.
5. On the **Instance Details** page, provide the following values:

1. Select **Network** > **devassoc** VPC.
2. Select **Subnet** > **Private Subnet**.
3. Select **IAM Role** > **devassoc-webserver**.
4. Select **Advanced Details** ➢ **User Data** ➢ **As File**.

5. From the folder where you downloaded the samples for this guide, select **Choose File** > **chapter-02/server-polly.txt**.

6. Choose **Next: Add Storage**.
7. Choose **Next: Add Tags**.
8. Select **Tags** > **Add tag** and set the following values:

1. For **Key**, enter **Name**.
2. For **Value**, enter **private-instance**.

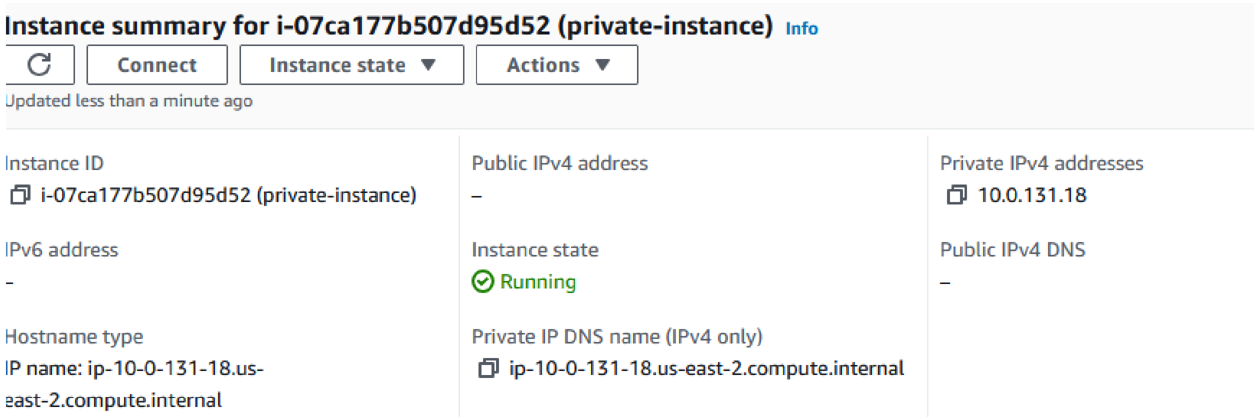
9. Choose **Next: Configure Security Group**.
10. For **Security Group**, set the following values:

1. For **Security group name**, enter **open-http-ssh**.
2. For **Description**: enter **HTTP and SSH from Anywhere**.

11. For the SSH rule, select **Source** > **Anywhere**.
12. Select **Add Rule** and then configure the second rule:

1. For **Type**, select HTTP.
2. For **Source**, select Anywhere.

13. Choose **Review and Launch**.
14. Choose **Launch**.
15. Under **Select a key pair**, choose **devassoc** and select the check box acknowledging that you have access to the key pair.



Instance summary for i-07ca177b507d95d52 (private-instance) Info

Updated less than a minute ago

| | | |
|---|---|---|
| Instance ID | Public IPv4 address | Private IPv4 addresses |
| i-07ca177b507d95d52 (private-instance) | – | 10.0.131.18 |
| IPv6 address | Instance state | Public IPv4 DNS |
| – | Running | – |
| Hostname type | Private IP DNS name (IPv4 only) | |
| IP name: ip-10-0-131-18.us-east-2.compute.internal | ip-10-0-131-18.us-east-2.compute.internal | |

## Exercise 2.8: Make Requests to Private Instance

In this exercise, you will explore connectivity to the private instance.

1. From your web browser, navigate to the private IP of the instance. Though the security group is open to requests from anywhere, this will fail because the private IP address is not routable over the internet.

2. Select **Services** > **EC2**.
3. From the list of instances, select **webserver**.
4. Select **Connect** and then follow the directions to establish an SSH connection.
5. From within the SSH session, make an HTTP request to the private server with curl. Replace the variable private-ip-address with the private IP address of **private-instance** address that you copied earlier.

curl private-ip-address

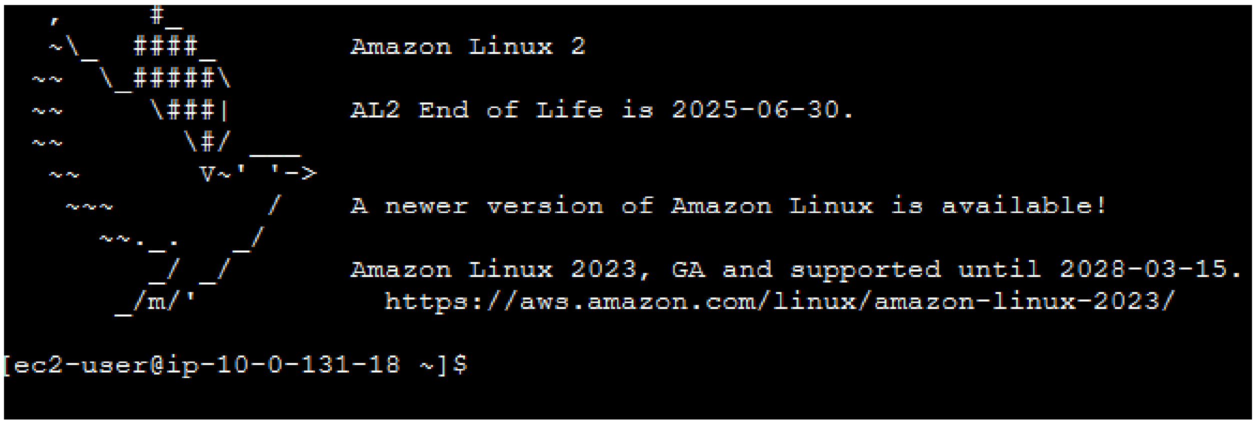
6. Download the MP3 audio from the **private-instance** to **webserver** using curl as follows:

curl private-ip-address/instance.mp3 --output instance.mp3

7. Make the file available for download from **webserver**:

sudo cp instance.mp3 /var/www/html/instance.mp3

8. In your web browser, enter the following address. Substitute public-ip-of- webserver with the public IPv4 address of **webserver**, and listen to the MP3.



```
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-131-18 ~]$
```

Downloaded the mp3 file

2.9: Cloud 9 no longer exist for any account I create.

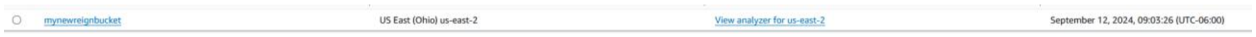
2.10.terminating environments such as webserver and private instance.

2.11 Deleted devassoc nat, all cloud instances, all security instances, only users remain now. Cleanup complete.

# Chapter 3: Hello Storage

## **Exercise 3.1:** Create an Amazon Simple Storage Service (Amazon S3) Bucket

Created an amazon bucket using the CLI in amazon rather than java SDK. Either way, you will use the code below as your import.



| | | | |
|---|---|---|---|
| mynewreignbucket | US East (Ohio) us-east-2 | View analyzer for us-east-2 | September 12, 2024, 09:03:26 (UTC-06:00) |

[

```
import java.io.IOException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.GetBucketLocationRequest;

public class CreateBucket {

  public static void main(String[] args) throws IOException {
    String clientRegion = "us-east";
    String bucketName = "davistestbucketformason";

    try {
      AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
          .withCredentials(new ProfileCredentialsProvider())
          .withRegion(clientRegion)
          .build();

      if (!s3Client.doesBucketExistV2(bucketName)) {
        // Because the CreateBucketRequest object doesn't specify a region, the
```

```
            // bucket is created in the region specified in the client.

            s3Client.createBucket(new CreateBucketRequest(bucketName));

            // Verify that the bucket was created by retrieving it and checking its location.

            String bucketLocation =
s3Client.getBucketLocation(new  GetBucketLocationRequest(bucketName));
            System.out.println("Bucket location: " + bucketLocation);
         }
      }
      catch(AmazonServiceException e) {
         // The call was transmitted successfully, but Amazon S3 couldn't process

         // it and returned an error response.

         e.printStackTrace();
      }
      catch(SdkClientException e) {
         // Amazon S3 couldn't be contacted for a response, or the client

         // couldn't parse the response from Amazon S3.

         e.printStackTrace();
      }
   }
}]
```

Replace the static variable values for clientRegion and bucketName. Note that bucket names must be unique across all of AWS. Make a note of these two values; you will use the same region and bucket name for the exercises that follow in this chapter.

## **Exercise 3.2:** Upload an object to a bucket

This code will upload an object to the bucket import java.io.File;

```
[
import java.io.IOException;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class UploadObject {

    public static void main(String[] args) throws IOException {
        String clientRegion = "*** Client region ***";
        String bucketName = "*** Bucket name ***";
        String stringObjKeyName = "*** String object key name ***";
        String fileObjKeyName = "*** File object key name ***";
        String fileName = "*** Path to file to upload ***";

        try {
            AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
                    .withRegion(clientRegion)
                    .withCredentials(new ProfileCredentialsProvider())
                    .build();

            // Upload a text string as a new object.
            s3Client.putObject(bucketName, stringObjKeyName, "Uploaded String Object");


            // Upload a file as a new object with ContentType and title specified.


            PutObjectRequest request = new PutObjectRequest(bucketName, fileObjKeyName, new
File(fileName));

            ObjectMetadata metadata = new ObjectMetadata();

            metadata.setContentType("plain/text");

            metadata.addUserMetadata("x-amz-meta-title", "someTitle");

            request.setMetadata(metadata);

            s3Client.putObject(request);

        }

        catch(AmazonServiceException e) {

            // The call was transmitted successfully, but Amazon S3 couldn't process


            // it, so it returned an error response.


            e.printStackTrace();

        }

        catch(SdkClientException e) {

            // Amazon S3 couldn't be contacted for a response, or the client


            // couldn't parse the response from Amazon S3.


            e.printStackTrace();

        }

    }

}


]
```

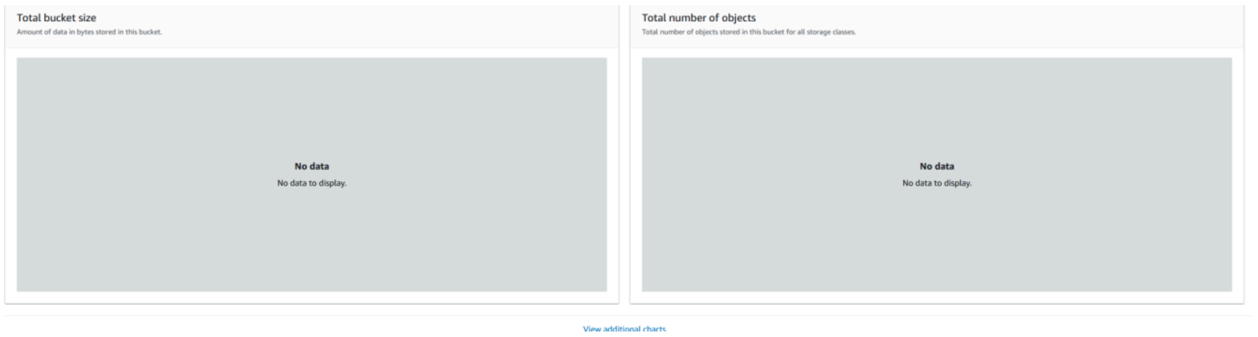
## **Exercise 3.3:** Emptying and Deleting a Bucket

Enter the following code in your preferred development environment for Java:

```
[

import com.amazonaws.AmazonServiceException;
import com.amazonaws.SdkClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;

public class DeleteBucket {

  public static void main(String[] args) {
    String clientRegion = "*** Client region ***";
    String bucketName = "*** Bucket name ***";

    try {
      AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
          .withCredentials(new ProfileCredentialsProvider())
          .withRegion(clientRegion)
          .build();

      // Delete all objects from the bucket. This is sufficient

      // for unversioned buckets. For versioned buckets, when you attempt to delete objects, Amazon S3
inserts
```

```
// delete markers for all objects, but doesn't delete the object versions.

// To delete objects from versioned buckets, delete all of the object versions before deleting

// the bucket (see below for an example).

ObjectListing objectListing = s3Client.listObjects(bucketName);
while (true) {
    Iterator<S3ObjectSummary> objIter = objectListing.getObjectSummaries().iterator();
    while (objIter.hasNext()) {
        s3Client.deleteObject(bucketName, objIter.next().getKey());
    }

    // If the bucket contains many objects, the listObjects() call

    // might not return all of the objects in the first listing. Check to

    // see whether the listing was truncated. If so, retrieve the next page of objects

    // and delete them.

    if (objectListing.isTruncated()) {
        objectListing = s3Client.listNextBatchOfObjects (objectListing);
    } else {
        break;
    }
}

// Delete all object versions (required for versioned buckets).
```

```
            VersionListing versionList = s3Client.listVersions(new
ListVersionsRequest().withBucketName(bucketName));

            while (true) {

                Iterator<S3VersionSummary> versionIter = versionList.getVersionSummaries().iterator();

                while (versionIter.hasNext()) {

                    S3VersionSummary vs = versionIter.next();

                    s3Client.deleteVersion(bucketName, vs.getKey(), vs.getVersionId());

                }


                if (versionList.isTruncated()) {

                    versionList = s3Client.listNextBatchOfVersions (versionList);

                } else {

                    break;

                }

            }


            // After all objects and object versions are deleted, delete the bucket.


            s3Client.deleteBucket(bucketName);

        }

        catch(AmazonServiceException e) {

            // The call was transmitted successfully, but Amazon S3 couldn't process


            // it, so it returned an error response.


            e.printStackTrace();

        }

        catch(SdkClientException e) {

            // Amazon S3 couldn't be contacted for a response, or the client couldn't


            // parse the response from Amazon S3.
            e.printStackTrace();
        }
    }
}
]
```


Total bucket size
Amount of data in bytes stored in this bucket.
No data
No data to display.
Total number of objects
Total number of objects stored in this bucket for all storage classes.
No data
No data to display.
View additional charts

# Chapter 4: Hello Databases

## Exercise 4.1: Create a Security Group for the Database Tier on RDS

Before you can create your first Amazon RDS database, you must create a security group so that you can allow traffic from your development server to communicate with the database tier. To do this, you must use an Amazon EC2 client to create the security group. Security groups are a component of the Amazon EC2 service, even though you can use them as part of Amazon RDS to secure your database tier.

To create the security group, run the following script:

```
# Excercise 4.1

import boto3

import json

import datetime


# Let's create some variables we'll use throughout these Excercises in Chapter 4

# NOTE: Here we are using a CIDR range for incoming traffic. We have set it to
   0.0.0.0/0 which means

# ANYONE on the internet can access your database if they have the username and
   the password

# If possible, specify you're own CIDR range. You can figure out your CIDR range
   by visiting the following link

# https://www.google.com/search?q=what+is+my+ip

# In the variable don't forget to add /32!

# If you aren't sure, leave it open to the world


# Variables

sg_name = 'rds-sg-dev-demo'

sg_description = 'RDS Security Group for AWS Dev Study Guide'

my_ip_cidr = '0.0.0.0/0'
```

```
# Create the EC2 Client to create the Security Group for your Database
ec2_client = boto3.client('ec2')


# First we need to create a security group
response = ec2_client.create_security_group(
    Description=sg_description,
    GroupName=sg_name)
print(json.dumps(response, indent=2, sort_keys=True))
# Now add a rule for the security group
response = ec2_client.authorize_security_group_ingress(
    CidrIp=my_ip_cidr,
    FromPort=3306,
    GroupName=sg_name,
    ToPort=3306,
    IpProtocol='tcp'
    )
print("Security Group should be created! Verify this in the AWS Console.")
```

After running the Python code, verify that the security group was created successfully from the AWS Management Console. You can find this confirmation under the VPC or Amazon EC2 service.

## Exercise 4.2: Spin Up the MariaDB Database Instance

Use the Python SDK to spin up your MariaDB database hosted on Amazon RDS.

To spin up the MariaDB database, run the following script and update the Variables section to meet your needs:

```
# Excercise 4.2
import boto3
import json
import datetime


# Just a quick helper function for date time conversions, in case you want to
```

```
   print the raw JSON
def date_time_converter(o):
   if isinstance(o, datetime.datetime):
      return o.__str__()

# Variables
sg_name = 'rds-sg-dev-demo'
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
admin_email = 'myemail@myemail.com'
sg_id_number = ''
rds_endpoint = ''

# We need to get the Security Group ID Number to use in the creation of the RDS
   Instance
ec2_client = boto3.client('ec2')
response = ec2_client.describe_security_groups(
   GroupNames=[
      sg_name
   ])

sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])
sg_id_number = sg_id_number.replace('"','')

#  Create the client for Amazon RDS
rds_client = boto3.client('rds')
```

```
# This will create our MariaDB Database
# NOTE: Here we are hardcoding passwords for simplicity and testing purposes
  only! In production
# you should never hardcode passwords in configuration files/code!
# NOTE: This will create an MariaDB Database. Be sure to remove it when you are
  done.
response = rds_client.create_db_instance(
  DBInstanceIdentifier=rds_identifier,
  DBName=db_name,
  DBInstanceClass='db.t2.micro',
  Engine='mariadb',
  MasterUsername='masteruser',
  MasterUserPassword='mymasterpassw0rd1!',
  VpcSecurityGroupIds=[
    sg_id_number
  ],
  AllocatedStorage=20,
  Tags=[
    {
      'Key': 'POC-Email',
      'Value': admin_email
    },
    {
      'Key': 'Purpose',
      'Value': 'AWS Developer Study Guide Demo'
    }
  ]
)
```

```
# We need to wait until the DB Cluster is up!
print('Creating the RDS instance. This may take several minutes...')
waiter = rds_client.get_waiter('db_instance_available')
waiter.wait(DBInstanceIdentifier=rds_identifier)


print('Okay! The Amazon RDS Database is up!')
```

After the script has executed, the following message is displayed:

```
Creating the RDS instance. This may take several minutes.
```

After the Amazon RDS database instance has been created successfully, the following confirmation is displayed:

```
Okay! The Amazon RDS Database is up!
```

You can also view these messages from the Amazon RDS console.

## Exercise 4.3: Obtain the Endpoint Value for the Amazon RDS Instance

Before you can start using the Amazon RDS instance, you must first specify your endpoint. In this exercise, you will use the Python SDK to obtain the value.

To obtain the Amazon RDS endpoint, run the following script:

```
# Exercise 4.3
import boto3
import json
import datetime



# Just a quick helper function for date time conversions, in case you want to    print the raw JSON
def date_time_converter(o):
    if isinstance(o, datetime.datetime):
        return o.__str__()


# Variables
rds_identifier = 'my-rds-db'
```

```
#  Create the client for Amazon RDS
rds_client = boto3.client('rds')


print("Fetching the RDS endpoint...")
response = rds_client.describe_db_instances(
   DBInstanceIdentifier=rds_identifier
)


rds_endpoint = json.dumps(response['DBInstances'][0]['Endpoint']['Address'])
rds_endpoint = rds_endpoint.replace('"','')
print('RDS Endpoint: ' + rds_endpoint)
```

After running the Python code, the following status is displayed:

```
Fetching the RDS endpoint.. RDS Endpoint:<endpoint_name>
```

If the endpoint is not returned, from the AWS Management Console, under the RDS service, verify that your Amazon RDS database instance was created.

## Exercise 4.4: Create a SQL Table and Add Records to It

You now have all the necessary information to create your first SQL table by using Amazon RDS. In this exercise, you will create a SQL table and add a couple of records. Remember to update the variables for your specific environment.

To update the variables, run the following script:

```
# Exercise 4.4
import boto3
import json
import datetime
import pymysql as mariadb


# Variables
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
```

```
user_name = 'masteruser'

user_password = 'mymasterpassw0rd1!'

rds_endpoint = 'my-rds-db.****.us-east-1.rds.amazonaws.com'


# Step 1 - Connect to the database to create the table

db_connection = mariadb.connect(host=rds_endpoint, user=user_name,

    password=user_password, database=db_name)

cursor = db_connection.cursor()

try:

    cursor.execute("CREATE TABLE Users (user_id INT NOT NULL AUTO_INCREMENT,

user_fname VARCHAR(100) NOT NULL, user_lname VARCHAR(150) NOT NULL, user_

email VARCHAR(175) NOT NULL, PRIMARY KEY (`user_id`))")

    print('Table Created!')

except mariadb.Error as e:

    print('Error: {}'.format(e))

finally:

    db_connection.close()


# Step 2 - Connect to the database to add users to the table

db_connection = mariadb.connect(host=rds_endpoint, user=user_name,

  password=user_password, database=db_name)

cursor = db_connection.cursor()

try:

    sql = "INSERT INTO `Users` (`user_fname`, `user_lname`, `user_email`) VALUES (%s, %s,
%s)"

    cursor.execute(sql, ('CJ', 'Smith', 'casey.smith@somewhere.com'))

    cursor.execute(sql, ('Casey', 'Smith', 'sam.smith@somewhere.com'))

    cursor.execute(sql, ('No', 'One', 'no.one@somewhere.com'))

# No data is saved unless we commit the transaction!
```

```
    db_connection.commit()
    print('Inserted Data to Database!')
except mariadb.Error as e:
    print('Error: {}'.format(e))
    print('Sorry, something has gone wrong!')
finally:
    db_connection.close()
```

After running the Python code, the following confirmation is displayed:

```
Table Created! Inserted Data to the Database!
```

Your Amazon RDS database now has some data stored in it.

## Exercise 4.5: Query the Items in the SQL Table

After adding data to your SQL database, in this exercise you will be able to read or query the items in the *Users* table.

To read the items in the SQL table, run the following script:

```
# Exercise 4.5
import boto3
import json
import datetime
import pymysql as mariadb

# Variables
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
rds_endpoint = 'my-rds-db.*****.us-east-1.rds.amazonaws.com'
```

```
db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
password=user_password, database=db_name)

cursor = db_connection.cursor()

try:

  sql = "SELECT * FROM `Users`"

  cursor.execute(sql)

  query_result = cursor.fetchall()

  print('Querying the Users Table...')

  print(query_result)

except mariadb.Error as e:

  print('Error: {}'.format(e))

  print('Sorry, something has gone wrong!')

finally:

  db_connection.close()
```

After running the Python code, you will see the three records that you inserted in the previous exercise.

## Exercise 4.6: Remove Amazon RDS Database and Security Group

You've created an Amazon RDS DB instance and added data to it. In this exercise, you will remove a few resources from your account. Remove the Amazon RDS instance first.

To remove the Amazon RDS instance and the security group, run the following script:

```
# Exercise 4.6

import boto3

import json

import datetime


# Variables

rds_identifier = 'my-rds-db'

sg_name = 'rds-sg-dev-demo'

sg_id_number = ''
```

```
# Create the client for Amazon RDS

rds_client = boto3.client('rds')


# Delete the RDS Instance

response = rds_client.delete_db_instance(

  DBInstanceIdentifier=rds_identifier,

  SkipFinalSnapshot=True)


print('RDS Instance is being terminated...This may take several minutes.')


waiter = rds_client.get_waiter('db_instance_deleted')

waiter.wait(DBInstanceIdentifier=rds_identifier)


# We must wait to remove the security groups until the RDS database has been     deleted, this is a
dependency.

print('The Amazon RDS database has been deleted. Removing Security Groups')



# Create the client for Amazon EC2 SG

ec2_client = boto3.client('ec2')


# Get the Security Group ID Number

response = ec2_client.describe_security_groups(

  GroupNames=[

    sg_name

  ])

sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])

sg_id_number = sg_id_number.replace('"','')
```

```
# Delete the Security Group!

response = ec2_client.delete_security_group(
    GroupId=sg_id_number
    )


print('Cleanup is complete!')
```

After running the Python code, the following message is displayed:

```
Cleanup is complete!
```

The Amazon RDS database and the security group are removed. You can verify this from the AWS Management Console.

## Exercise 4.7: Create an Amazon DynamoDB Table

Amazon DynamoDB is a managed NoSQL database. One major difference between DynamoDB and Amazon RDS is that DynamoDB doesn't require a server that is running in your VPC, and you don't need to specify an instance type. Instead, create a table.

To create the table, run the following script:

```
# Exercise 4.7

import boto3

import json

import datetime


dynamodb_resource = boto3.resource('dynamodb')


table = dynamodb_resource.create_table(
    TableName='Users',
    KeySchema=[
        {
            'AttributeName': 'user_id',
            'KeyType': 'HASH'
        },
    {
      'AttributeName': 'user_email',
      'KeyType': 'RANGE'
    }
  ],
  AttributeDefinitions=[
    {
      'AttributeName': 'user_id',
      'AttributeType': 'S'
    },
    {
      'AttributeName': 'user_email',
      'AttributeType': 'S'
    }
  ],
  ProvisionedThroughput={
    'ReadCapacityUnits': 5,
    'WriteCapacityUnits': 5
  }
)

print("The DynamoDB Table is being created, this may take a few minutes...")
table.meta.client.get_waiter('table_exists').wait(TableName='Users')
print("Table is ready!")
```

After running the Python code, the following message is displayed:

```
Table is ready!
```

From the AWS Management Console, under DynamoDB, verify that the table was created.

## Exercise 4.8: Add Users to the Amazon DynamoDB Table

With DynamoDB, there are fewer components to set up than there are for Amazon RDS. In this exercise, you'll add users to your table. Experiment with updating and changing some of the code to add multiple items to the database.

To add users to the DynamoDB table, run the following script: # Exercise 4.8

```
import boto3

import json

import datetime

# In this example we are not using uuid; however, you could use this to autogenerate your user IDs.

# i.e. str(uuid.uuid4())

import uuid


# Create a DynamoDB Resource

dynamodb_resource = boto3.resource('dynamodb')

table = dynamodb_resource.Table('Users')


# Write a record to DynamoDB

response = table.put_item(

   Item={

      'user_id': '1234-5678',

      'user_email': 'someone@somewhere.com',

      'user_fname': 'Sam',

      'user_lname': 'Samuels'

   }

)


# Just printing the raw JSON response, you should see a 200 status code

print(json.dumps(response, indent=2, sort_keys=True))
```

After running the Python code, you receive a 200 HTTP Status Code from AWS. This means that the user record has been added.

From the AWS Management Console, under DynamoDB, review the table to verify that the user record was added.

## Exercise 4.9: Look Up a User in the Amazon DynamoDB Table

In this exercise, you look up the one user you've added so far.

To look up users in the DynamoDB table, run the following script:

```
# Exercise 4.9

import boto3

from boto3.dynamodb.conditions import Key

import json

import datetime


# Create a DynamoDB Resource

dynamodb_resource = boto3.resource('dynamodb')

table = dynamodb_resource.Table('Users')


# Query a some data

response = table.query(

  KeyConditionExpression=Key('user_id').eq('1234-5678')

)


# Print the data out!

print(json.dumps(response['Items'], indent=2, sort_keys=True))
```

After running the Python code, the query results are returned in JSON format showing a single user.

## Exercise 4.10: Write Data to the Table as a Batch Process

In this exercise, you will write data to the table through a batch process.

To write data using a batch process, run the following script:

```
# Exercise 4.10
import boto3
import json
import datetime
import uuid

# Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

# Generate some random data
with table.batch_writer() as user_data:
    for i in range(100):
        user_data.put_item(
            Item={
                'user_id': str(uuid.uuid4()),
                'user_email': 'someone' + str(i) + '@somewhere.com',
                'user_fname': 'User' + str(i),
                'user_lname': 'UserLast' + str(i)
            }
        )
        print('Writing record # ' + str(i+1) + ' to DynamoDB Users Table')
    print('Done!')
```

After running the Python code, the last few lines read as follows:

```
Writing record # 300 to DyanmoDB Users Table Done!
```

From the AWS Management Console, under DynamoDB Table, verify that the users were written to the table.

## Exercise 4.11: Scan the Amazon DynamoDB Table

In this exercise, you will scan the entire table.

To scan the table, run the following script:

```
# Exercise 4.11
import boto3
import json
import datetime
import uuid

# Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

# Let's do a scan!
response = table.scan()

print('The total Count is: ' + json.dumps(response['Count']))
print(json.dumps(response['Items'], indent=2, sort_keys=True))
```

As you learned in this chapter, scans return the entire dataset located in the table. After running the script, all of the users are returned.

## Exercise 4.12: Remove the Amazon DynamoDB Table

In this exercise, you will remove the DynamoDB table that you created in Exercise 4.7.

To remove the table, run the following script:

```
# Exercise 4.12
import boto3
import json
import datetime
import uuid
```

```
# Create a DynamoDB Resource

dynamodb_client = boto3.client('dynamodb')


# Delete the Table

response = dynamodb_client.delete_table(TableName='Users')

print(json.dumps(response, indent=2, sort_keys=True))
```

The DynamoDB table is deleted, or it is in the process of being deleted. Verify the deletion from the AWS Management Console, under the DynamoDB service.

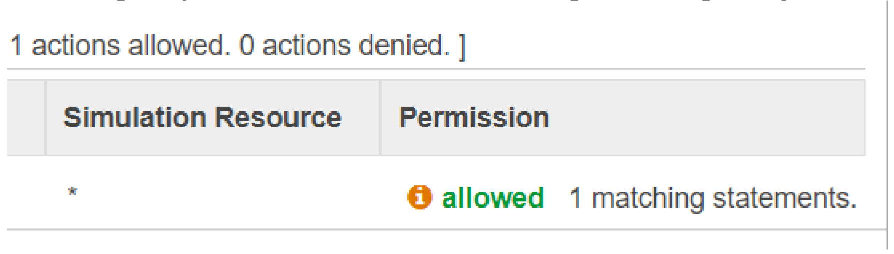
# Chapter 5: Encryption on AWS

## **Exercise 5.1:** Configure an Amazon S3 Bucket to Deny Unencrypted Uploads

We are encrypting a bucket we are making in S3 in AWS. First create the bucket, in this case it will be named “mytestbucketmdavis” located in a default region. Below is the script we are going to add as a policy for the bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyIncorrectEncryption",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::mytestbucketmdavis/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "AES256"
        }
      }
    },
    {
      "Sid": "DenyMissingEncryption",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::mytestbucketmdavis/*",
      "Condition": {
        "Null": {
          "s3:x-amz-server-side-encryption": "true"
        }
      }
    }
  ]
}
```

Then we run the policy simulation, below is the setup. We use put object action on AmazonS3

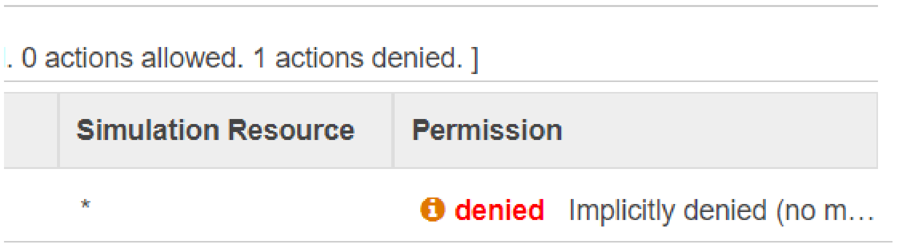


1 actions allowed. 0 actions denied. ]

| | Simulation Resource | Permission |
|---|---|---|
| | * | allowed 1 matching statements. |

service.

Now without the s3:x-amz-server-side-encryption enabled in the bucket's policy, it should be denied.



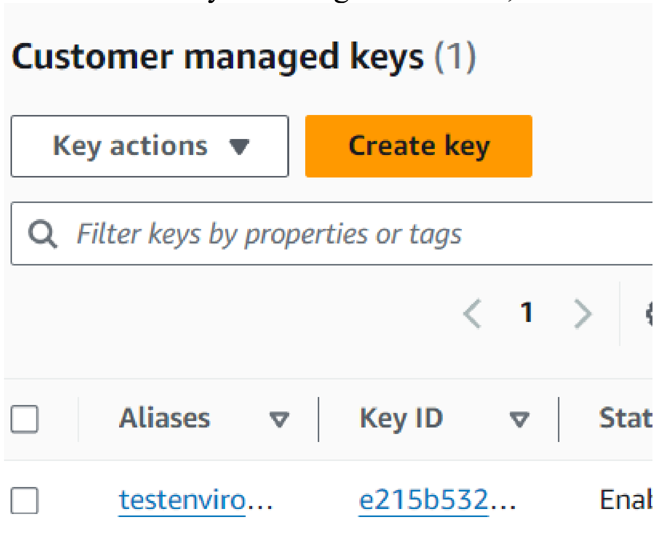
. 0 actions allowed. 1 actions denied. ]

| | Simulation Resource | Permission |
|---|---|---|
| | * | denied Implicitly denied (no m… |

## Exercise 5.2: Create and disable an AWS Key Management Service

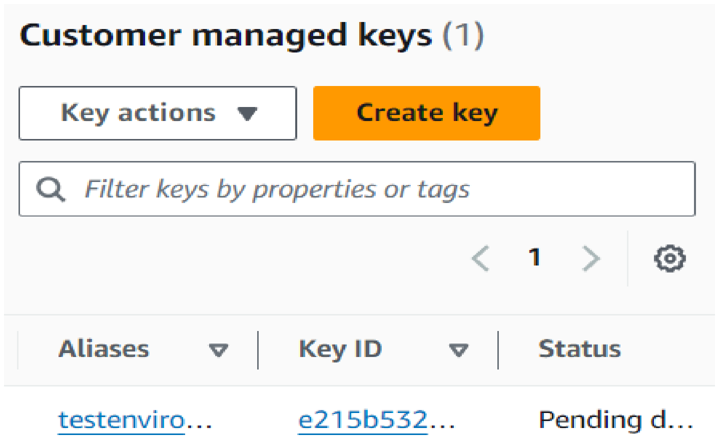
In this section, we will be covering creating a key and disabling it using the KMS system provided by AWS.

First create a key and assign it to a user, for test example, is has been assigned to a dummy user.



| | Aliases | Key ID | Stat |
|---|---|---|---|
| | testenviro… | e215b532… | Enal |

Now, all you have to do to disable this key is go under key actions, with the key selected, and disable. For this scenario, we are scheduling deletion since we are done using this key and will be making a new one for the next section.



| Aliases | Key ID | Status |
|---|---|---|
| testenviro… | e215b532… | Pending d… |

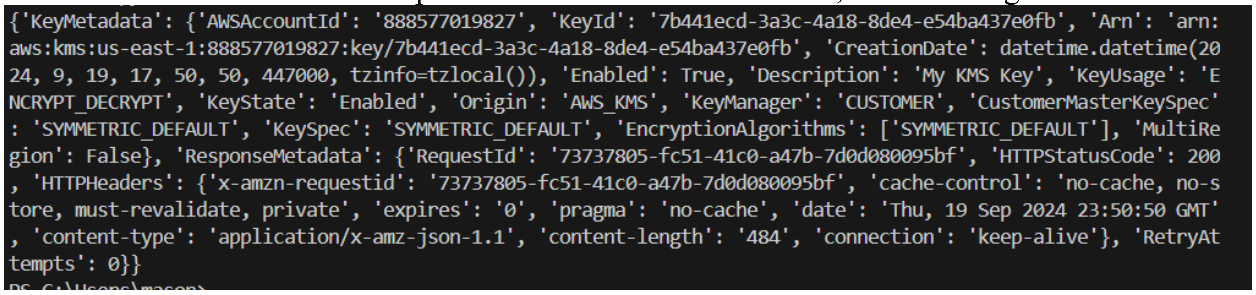
## **Exercise 5.3:** Create an AWS KMS Customer Master Key with Python SDK

In this section we will be creating a customer master key (CMK) using python.

To create the CMK, run the following script, this can only encrypt and decrypt with what functions we have assigned.

```
import boto3
import json

kms_client = boto3.client('kms', region_name='us-west-1')

response = kms_client.create_key(
    Description='My KMS Key',
    KeyUsage='ENCRYPT_DECRYPT',
    Origin='AWS_KMS',
    Tags=[
        {
            'TagKey': 'KeyPurpose',
            'TagValue': 'dev-on-aws-key'
        },
    ]
)

print(response)
```

This should be the meta data response back from the AWS server, or something like this:

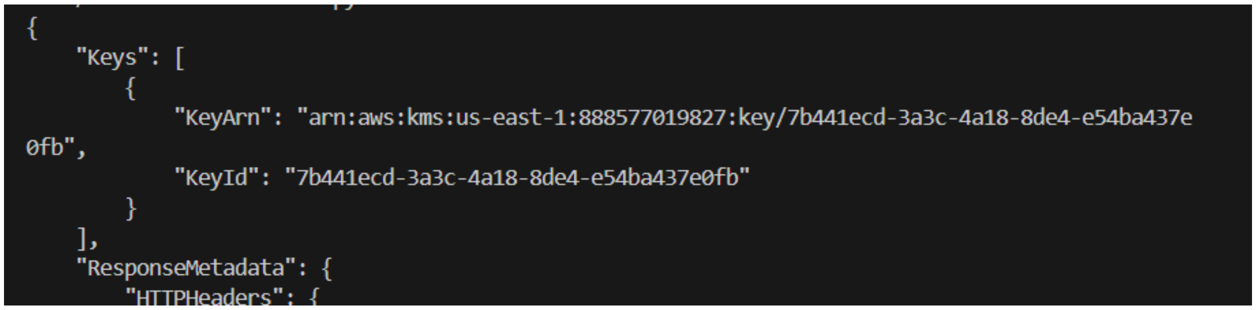


```
{'KeyMetadata': {'AWSAccountId': '888577019827', 'KeyId': '7b441ecd-3a3c-4a18-8de4-e54ba437e0fb', 'Arn': 'arn:
aws:kms:us-east-1:888577019827:key/7b441ecd-3a3c-4a18-8de4-e54ba437e0fb', 'CreationDate': datetime.datetime(20
24, 9, 19, 17, 50, 50, 447000, tzinfo=tzlocal()), 'Enabled': True, 'Description': 'My KMS Key', 'KeyUsage': 'E
NCRYPT_DECRYPT', 'KeyState': 'Enabled', 'Origin': 'AWS_KMS', 'KeyManager': 'CUSTOMER', 'CustomerMasterKeySpec'
: 'SYMMETRIC_DEFAULT', 'KeySpec': 'SYMMETRIC_DEFAULT', 'EncryptionAlgorithms': ['SYMMETRIC_DEFAULT'], 'MultiRe
gion': False}, 'ResponseMetadata': {'RequestId': '73737805-fc51-41c0-a47b-7d0d080095bf', 'HTTPStatusCode': 200
, 'HTTPHeaders': {'x-amzn-requestid': '73737805-fc51-41c0-a47b-7d0d080095bf', 'cache-control': 'no-cache, no-s
tore, must-revalidate, private', 'expires': '0', 'pragma': 'no-cache', 'date': 'Thu, 19 Sep 2024 23:50:50 GMT'
, 'content-type': 'application/x-amz-json-1.1', 'content-length': '484', 'connection': 'keep-alive'}, 'RetryAt
tempts': 0}}
```

Now that the key is created, we are now going to list the CMKS and describe the available keys, using this script. Remember to change region name to what your configuration file in .aws is set to.

```
import boto3
import json
from botocore import ClientError
import logging
# List the KMS Keys by ID
kms_client = boto3.client('kms', region_name=your-region-0')
try:
    response = kms_client.list_keys()
except ClientError as e:
    logging.error(e)

print(json.dumps(response, indent=4, sort_keys=True))
```

You should get the following, other than the actual key value since that will change, along with some meta data following the key information



```
{
    "Keys": [
        {
            "KeyArn": "arn:aws:kms:us-east-1:888577019827:key/7b441ecd-3a3c-4a18-8de4-e54ba437e
0fb",
            "KeyId": "7b441ecd-3a3c-4a18-8de4-e54ba437e0fb"
        }
    ],
    "ResponseMetadata": {
```

The following script is now going to be used to descripe the actual key itself using the keys meta data, along with all other information.

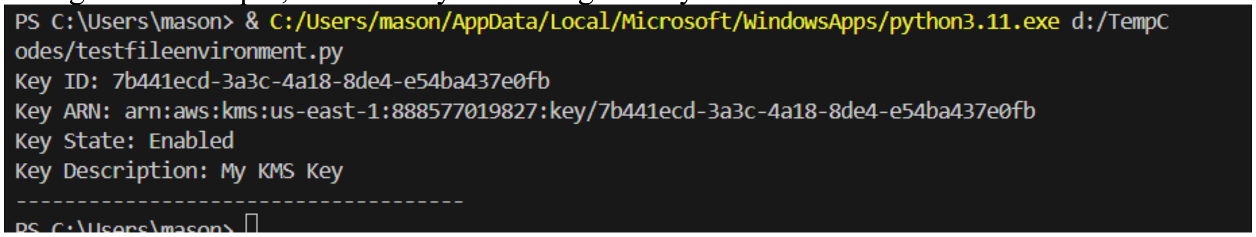
```
import boto3
import json
import logging
from botocore import ClientError
# List the KMS Keys by ID
kms_client = boto3.client('kms', region_name='us-west-1')

# Describe the Keys
```

```
for key in response['Keys']:
    try:
        key_info = kms_client.describe_key(KeyId=key['KeyArn'])
        key_id = key_info['KeyMetadata']['KeyId']
        key_arn = key_info['KeyMetadata']['Arn']
        key_state = key_info['KeyMetadata']['KeyState']
        key_description = key_info['KeyMetadata']['Description']
        print('Key ID: ' + key_id)
        print('Key ARN: ' + key_arn)
        print('Key State: ' + key_state)
        print('Key Description: ' + key_description)
        print('--------------------------------------')
    except ClientError as e:
        logging.error(e)
```

This code works on legacy systems, but no longer applies here. I have created a new section of code and it is currently working, another solution using updated libraries with the help of documentation by AWS. Below is what was run for the following input below the code.

```
import boto3
import json
import logging
from botocore.exceptions import ClientError

# Configure logging
logging.basicConfig(level=logging.ERROR)

# Initialize KMS client
kms_client = boto3.client('kms', region_name='us-east-1')

# List KMS keys
try:
    response = kms_client.list_keys()
    if 'Keys' in response:
        for key in response['Keys']:
            try:
                # Use KeyId instead of KeyArn
                key_info = kms_client.describe_key(KeyId=key['KeyId'])
                key_id = key_info['KeyMetadata']['KeyId']
                key_arn = key_info['KeyMetadata']['Arn']
                key_state = key_info['KeyMetadata']['KeyState']
                key_description = key_info['KeyMetadata'].get('Description', 'No description available')

                # Using f-strings for clean formatting
                print(f'Key ID: {key_id}')
                print(f'Key ARN: {key_arn}')
                print(f'Key State: {key_state}')
                print(f'Key Description: {key_description}')
                print('--------------------------------------')
            except ClientError as e:
                logging.error(e)
    else:
        print("No keys found.")
except ClientError as e:
    logging.error(e)
```

This gives this output, successfully describing the key.

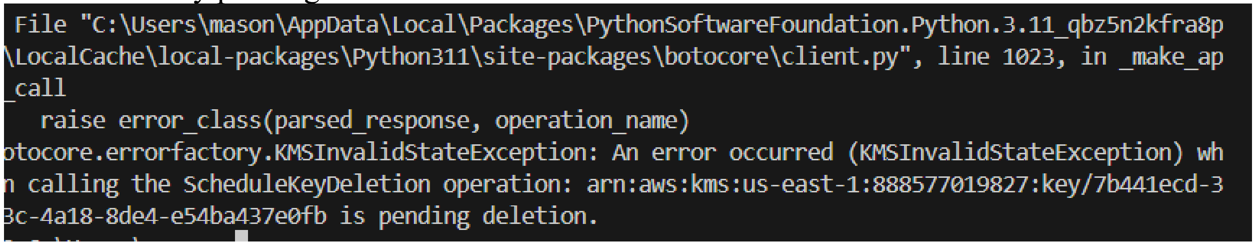


```
PS C:\Users\mason> & C:/Users/mason/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/TempC
odes/testfileenvironment.py
Key ID: 7b441ecd-3a3c-4a18-8de4-e54ba437e0fb
Key ARN: arn:aws:kms:us-east-1:888577019827:key/7b441ecd-3a3c-4a18-8de4-e54ba437e0fb
Key State: Enabled
Key Description: My KMS Key
-------------------------------------
PS C:\Users\mason> 
```

Finally, you can now delete the key with the following script grabbed from the book we have been using.

```
import boto3

kms_client = boto3.client('kms', region_name='us-west-1')
response = kms_client.schedule_key_deletion(
    KeyId=yourkeyid',
    PendingWindowInDays=7
)

print(response, indent=4, sort_keys=True)
```

Now, I ran this with an error, so deletion went through but since I had the wrong key id it did not print the output correctly. That's dangerous. But re running the script will give you the response that it is already pending deletion



```
 File "C:\Users\mason\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p
\LocalCache\local-packages\Python311\site-packages\botocore\client.py", line 1023, in _make_ap
_call
    raise error_class(parsed_response, operation_name)
otocore.errorfactory.KMSInvalidStateException: An error occurred (KMSInvalidStateException) wh
n calling the ScheduleKeyDeletion operation: arn:aws:kms:us-east-1:888577019827:key/7b441ecd-3
3c-4a18-8de4-e54ba437e0fb is pending deletion.
```

# Chapter 6: The Elastic Beanstalk

Preset up: The application I will be deploying aws docker sample application retrieved from https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/tutorials.html

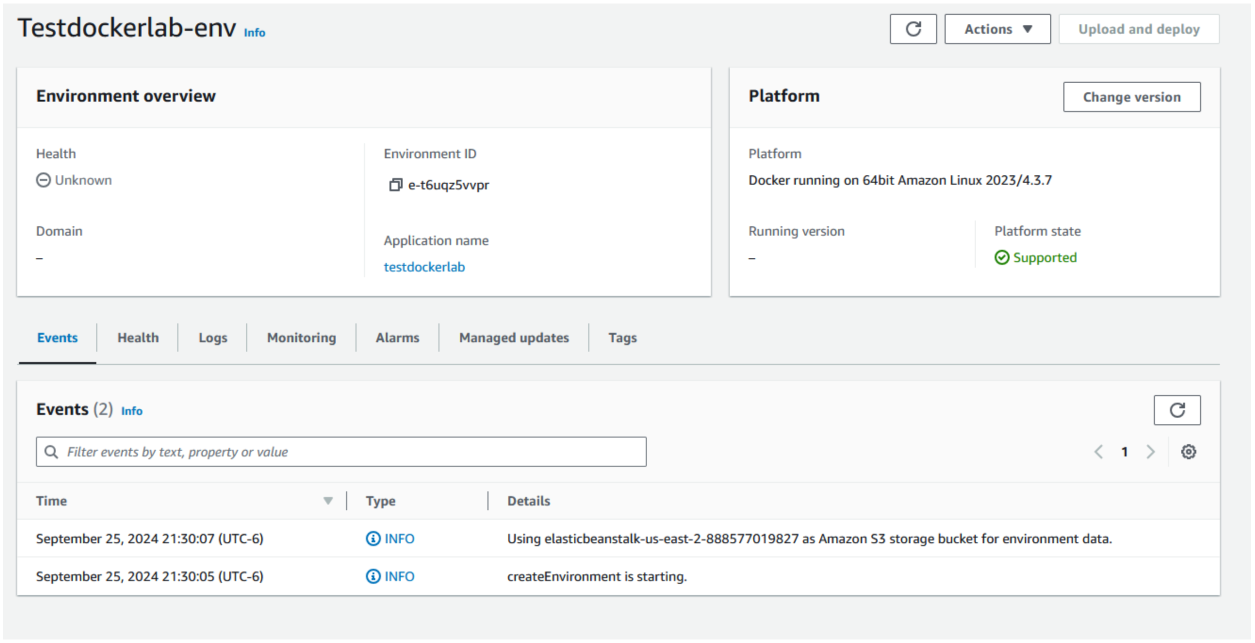
## **Exercise 6.1:** Deploy your Application

This step is getting the application into the elastic beanstalk. For this we will create a new application, name it whatever. In this scenario we are going to run docker platform, uploading our own code, which is the docker sample application, and using a single instance to keep the free tier eligible.

Create database with SQL through the creation menu as the test Data base. Make sure you are using a non-default VPC, so the database and application subnets are set up correctly. You can also set this up in the application deployment wizard.

No proxy server. Everything else is left default other than device, make sure you are using t3 micro to ensure free tier compliance.

Here is the environment creation screen when finished through the wizard.



Environment overview

Health: Unknown
Environment ID: e-t6uqz5vvpr
Domain: –
Application name: testdockerlab

Platform

Platform: Docker running on 64bit Amazon Linux 2023/4.3.7
Running version: –
Platform state: Supported

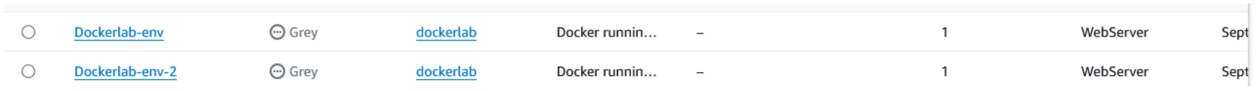
Events (2)

| Time | Type | Details |
|---|---|---|
| September 25, 2024 21:30:07 (UTC-6) | INFO | Using elasticbeanstalk-us-east-2-888577019827 as Amazon S3 storage bucket for environment data. |
| September 25, 2024 21:30:05 (UTC-6) | INFO | createEnvironment is starting. |

## Exercise 6.2: Deploy a Blue/Green Solution

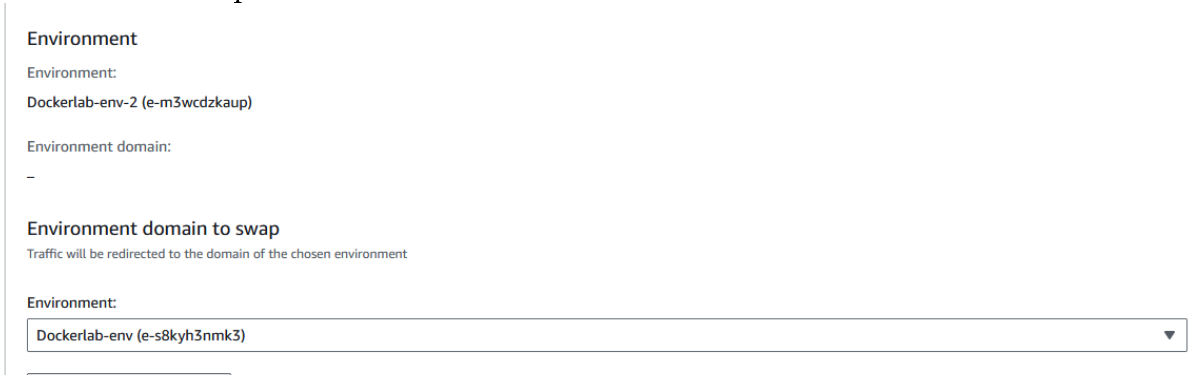
Deploy a green blue solution, this well help keep your database running during updates.

First duplicate the existing environment.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ○ | Dockerlab-env | Grey | dockerlab | Docker runnin… | – | 1 | WebServer | Sept |
| ○ | Dockerlab-env-2 | Grey | dockerlab | Docker runnin… | – | 1 | WebServer | Sept |

You will now swap env2 to env 1



Environment

Environment:
Dockerlab-env-2 (e-m3wcdzkaup)

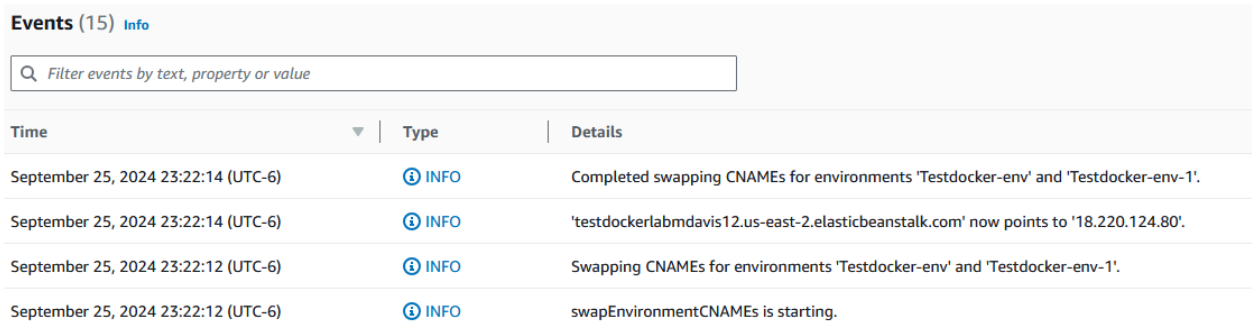
Environment domain:
–

Environment domain to swap

Traffic will be redirected to the domain of the chosen environment

Environment:
Dockerlab-env (e-s8kyh3nmk3)

This is the completed CNAME Swap of the environments.



Events (15) Info

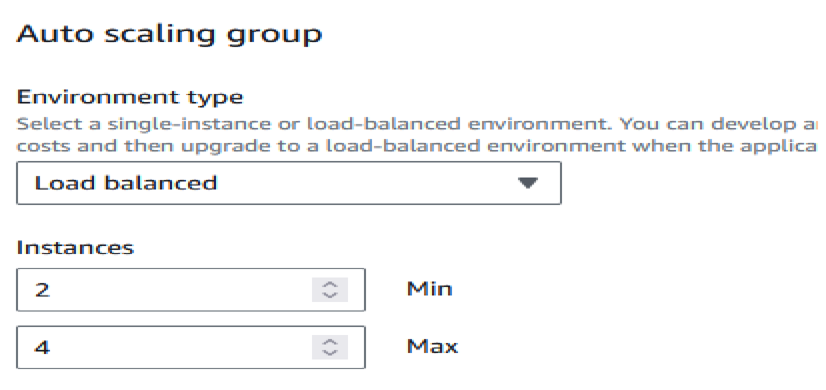
Filter events by text, property or value

| Time | Type | Details |
|---|---|---|
| September 25, 2024 23:22:14 (UTC-6) | INFO | Completed swapping CNAMEs for environments 'Testdocker-env' and 'Testdocker-env-1'. |
| September 25, 2024 23:22:14 (UTC-6) | INFO | 'testdockerlabmdavis12.us-east-2.elasticbeanstalk.com' now points to '18.220.124.80'. |
| September 25, 2024 23:22:12 (UTC-6) | INFO | Swapping CNAMEs for environments 'Testdocker-env' and 'Testdocker-env-1'. |
| September 25, 2024 23:22:12 (UTC-6) | INFO | swapEnvironmentCNAMEs is starting. |

## Exercise 6.3: Change your Environment Configuration

Here we are configuring the existing environments capacity configuration to load balanced, with max of 4 and min of 2.

You should have this:



Auto scaling group

Environment type

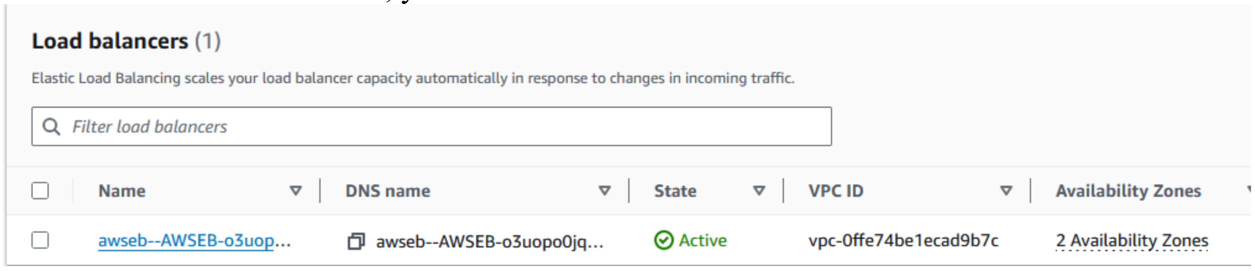
Select a single-instance or load-balanced environment. You can develop a costs and then upgrade to a load-balanced environment when the applica

Load balanced

Instances

| | |
|---|---|
| 2 | Min |
| 4 | Max |

In the EC2 Service dashboard, you should have load balancer active. As shown below.



Load balancers (1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

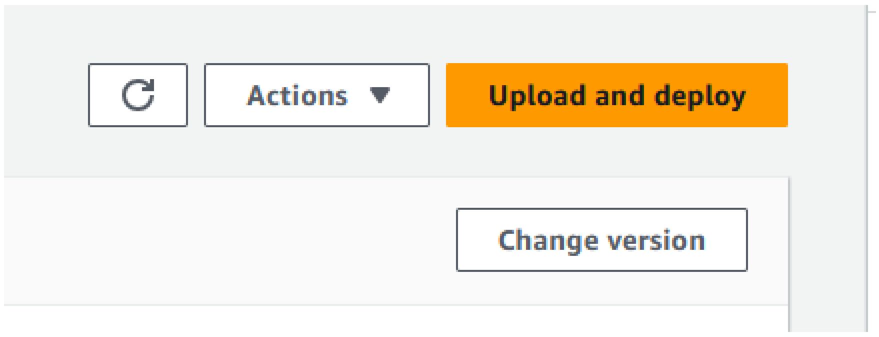
Filter load balancers

| | Name | DNS name | State | VPC ID | Availability Zones |
|---|---|---|---|---|---|
| ☐ | awseb--AWSEB-o3uop... | awseb--AWSEB-o3uopo0jq... | Active | vpc-0ffe74be1ecad9b7c | 2 Availability Zones |

## Exercise 6.4: Update an application on the Beanstalk

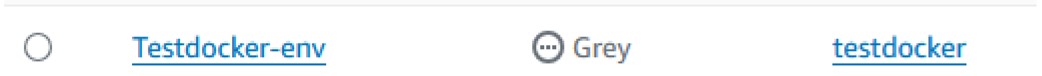
We are now going to update an application version using the AWS ELB. First create a second version of the application that matches the current configuration. First get the new application, then inside of the environemnt, you will hit upload and deploy.

Shown Below:



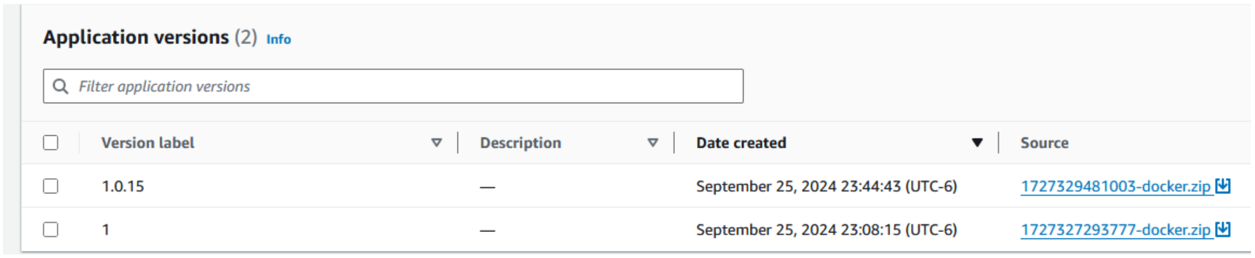
From there, select your new file to upload, and change the version.

You should now have the environment in grey when it is updating:



Testdocker-env Grey testdocker

When it is complete, under application versions inside of the environment, you should see close to the following to show versioning was successful for the deployment.



Application versions (2) Info

Filter application versions

| Version label | Description | Date created | Source |
|---|---|---|---|
| 1.0.15 | — | September 25, 2024 23:44:43 (UTC-6) | 1727329481003-docker.zip |
| 1 | — | September 25, 2024 23:08:15 (UTC-6) | 1727327293777-docker.zip |

# Chapter 7: Deployment as Code

It should be noted, Chapter 8 was originally dedicated to using amazon Code Commit Repo, but as of this day 10/7/24, it has since been depreciated. Nevertheless, this manual will contain the original plans for Code Commit since old companies who have been using it, will still be using it, only new members cannot use Code Commit. Photos will not be provided for Code Commit since it can no longer be done without access to an existing environment.

## Exercise 7.1: AWS CodeCommit Repo

This exercise demonstrates how to use AWS CodeCommit to submit, and merge pull requests to a repository.

1. Create an AWS CodeCommit repository with a name and description. You do not need to configure email notifications for repository events.
2. In the AWS CodeCommit console, select **Create File** to add a simple markdown file to test the repository.
3. Clone the repository to your local machine with HTTPS or SSH authentication.
4. Create a file locally, commit it to the repository, and push it to test the AWS CodeCommit.
5. Create a feature branch from the master branch in the repository.
6. Edit the file and commit the changes to the feature branch.
7. Use the AWS CodeCommit console to create a pull request. Use the master branch of the repository as the destination and the feature branch as the source.
8. After the pull request is successfully created, merge the changes from the feature branch with the master branch.

The pull request has been merged with the master branch, which can be confirmed by viewing the source code of the markdown file in the master branch.

## **Exercise 7.2:** Create an Application in AWS CodeDeploy

This exercise demonstrates how to use AWS CodeDeploy to perform an in-place deployment to Amazon EC2 instances in your account.

1. Create a new application in the AWS CodeDeploy console.

For the compute platform type, select **EC2 On-premises**.

2. Create a new deployment group for your application. Specify the following values:

**Deployment type** In-place

**Environment configuration** Amazon EC2 instances

**Tag group** Create a tag group that is easy to identify, such as a "Name" for the key, and "CodeDeployInstance" as the value.

**Load balancer** Clear the Enable load balancing check box.

3. Launch new Amazon EC2 instance.

Make sure to specify the tag value chosen in the previous step.

4. Download the sample application bundle to your local machine for future updates.

Sample application bundles for each operating system can be found using the following links:

**Windows Server** https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-windows.html

**Amazon Linux or Red Hat Enterprise Linux (RHEL)** https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-wordpress.html

5. Create a deployment group, and verify that the sample application deploys.
6. Update the application code, and submit a new deployment to the deployment group.
7. Verify your changes after the deployment completes.

## **Exercise 7.3:** Create an AWS CodeBuild Project

This exercise demonstrates how to use AWS CodeBuild to perform builds and the compilation of artifacts prior to deployment to Amazon EC2 instances.

1. Create an Amazon S3 bucket to hold artifacts.
2. Upload two or more arbitrary files to the bucket.
3. Use the AWS CodeBuild console to create a build project with the following settings:

**Project name** Provide a name of your choice.

**Source** Use Amazon S3.

**Bucket** Provide the name of the bucket you created.

**S3 object key** Provide the name of one of the objects you uploaded.

**Environment image** Select the Managed Image type.

**Operating system** Use Ubuntu.

**Runtime** Use Python.

**Runtime version** Select a version of your choice.

**Service role** Select New Service Role.

**Role name** Provide a name for your service role.

**Build specifications** Select Insert Build Commands.

**Build commands** Select Switch To Editor and enter the following. Replace the Amazon S3 object paths with paths to the objects you uploaded to your bucket.

```
version: 0.2


phases:
  build:
    commands:
      - aws s3 cp s3://yourbucket/file1 /tmp/file1
      - aws s3 cp s3://yourbucket/file2 /tmp/file2
artifacts:
  files:
    - /tmp/file1
    - /tmp/file2
```

**Artifact Type** Use Amazon S3.

**Bucket name** Select your Amazon S3 bucket.

**Artifacts packaging** Select Zip.

4. Save your build project.
5. Run your build project, and observe the output archive file created in your Amazon S3 bucket.

# Chapter 7: New Alternative

**Exercise 7.1.1:** A new route

Since as of this date, CodeCommit is no longer available to new customers, so we will be using GitHub with AWS integration.

**1. Create a GitHub Repository:**

- Go to your GitHub account, click **"New repository"**.
- Name your repository (e.g., my-github-repo) and add a description.
- Choose public or private depending on your preferences and click **"Create repository"**.

**2. Add a Markdown File to the Repository:**

- On the main page of the repository, click **"Add file"** and select **"Create new file"**.
- Name the file README.md, add some content (e.g., # My GitHub Repository), and commit the file directly to the main branch.

**3. Clone the Repository to Your Local Machine:**

- Use HTTPS or SSH to clone the repository. Get the URL from the repository's main page under the **"Code"** button.
- Clone using the following command:

```
git clone https://github.com/your-username/my-github-repo.git
```

**4. Create and Push a Local File to the Repository:**

- **Navigate to the repository folder on your local machine:**

```
cd my-github-repo
```

- Create a new file (e.g., new-file.md):

echo "# New File" > new-file.md

- **Add and commit the file to the local repository:**

git add new-file.md

git commit -m "Added new file"

- **Push the changes to GitHub:**

git push origin main

**5. Create a Feature Branch:**

- Create a new branch from the main branch:

git checkout -b feature-branch

- Edit the README.md file (e.g., adding ## Feature Branch Changes).
- Add and commit the changes:

git add README.md

git commit -m "Edited README in feature branch"

- Push the feature branch to GitHub:

git push origin feature-branch

**6. Create a Pull Request on GitHub:**

- Go to the repository page on GitHub.
- Click **"Pull requests"** and then **"New pull request"**.
- Choose the **main** branch as the base and **feature-branch** as the compare.
- Add a title and description for the pull request, then click **"Create pull request"**.

**7. Merge the Feature Branch into the Main Branch:**

- Once the pull request is reviewed and approved, click the **"Merge pull request"** button.
- Confirm the merge to combine the changes from feature-branch into main.

**8. Set Up AWS Integration (Optional for CI/CD):**

To integrate GitHub with AWS for automating deployments and managing continuous integration/delivery, you can use **AWS CodePipeline** or **GitHub Actions**:

**Option 1: Using AWS CodePipeline with GitHub:**

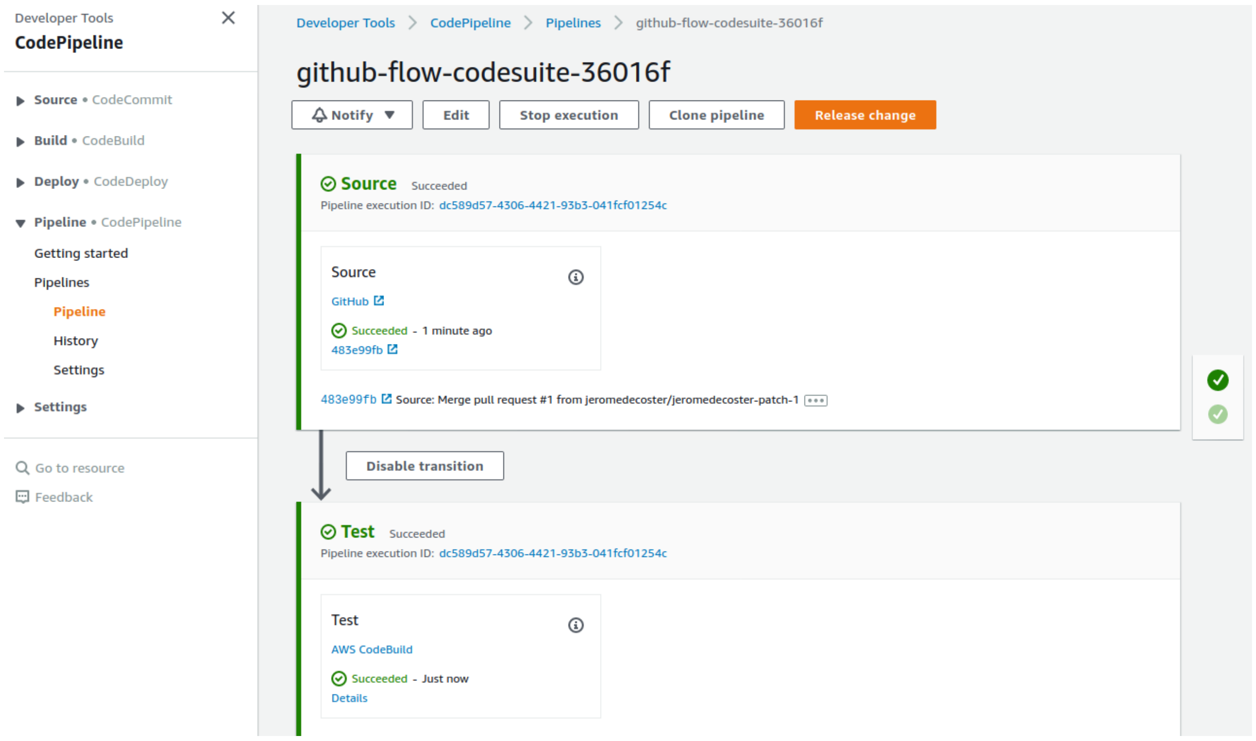
- Set up a **CodePipeline** in the AWS Management Console.
- During the source stage, choose **GitHub** as the source provider and link your GitHub repository.

- Set up build and deploy stages using **AWS CodeBuild** and/or **AWS CodeDeploy** to automatically build, test, and deploy your code changes to an AWS service (like EC2, Lambda, or ECS).

**Option 2: Using GitHub Actions to Deploy to AWS:**

- You can set up a **GitHub Actions workflow** to deploy code changes directly to AWS. Create a .github/workflows/deploy.yml file with the following template:

```
name: Deploy to AWS


on:
  push:
    branches:
      - main


jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2


      - name: Deploy to AWS
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-west-2


      - name: Deploy with AWS CLI
        run: aws s3 sync . s3://my-bucket-name
```



github-flow-codesuite-36016f

Source Succeeded
Pipeline execution ID: dc589d57-4306-4421-93b3-041fcf01254c

Source
GitHub
Succeeded - 1 minute ago
483e99fb

483e99fb Source: Merge pull request #1 from jeromedecoster/jeromedecoster-patch-1

Test Succeeded
Pipeline execution ID: dc589d57-4306-4421-93b3-041fcf01254c

Test
AWS CodeBuild
Succeeded - Just now
Details

# Chapter 8 : Infrastructure as Code

## **Exercise 8.1 :** Write your own AWS CloudFormation Template

For this chapter, we will be focusing on CloudFormation. For 8.1, we will be writing our own AWS CloudFormation Template.

First Create an Amazon S3 Bucket with a static website configuration that includes references to index and error documents(HTML Docs) like index.html and error.html. Once you create the bucket, enable static website at the bottom of properties, and under permissions, your code for the policy should be this

*"{ "Version": "2012-10-17",*

*"Statement": [*

*{ "Effect": "Allow",*

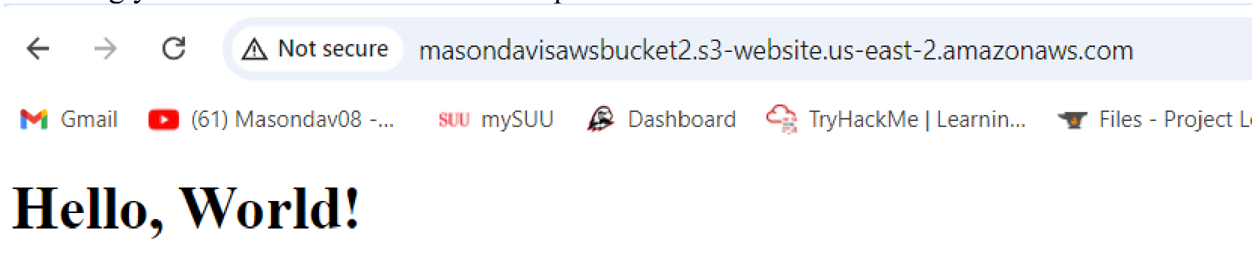
*"Principal": "*",*

*"Action": "s3:GetObject",*

*"Resource": "arn:aws:s3:::your-bucket-name/*" } ] }"*

DO NOT REMOVE THE *, that is the principal being called on for the resource, in this case, all. Of course, change in the future as needed.

Submit an index.html and an error.html as object into the bucket. You should now have this, assuming you used a basic hello world template.



Hello, World!

Your buckets URL is retrieved from the same place you set it as a static website.

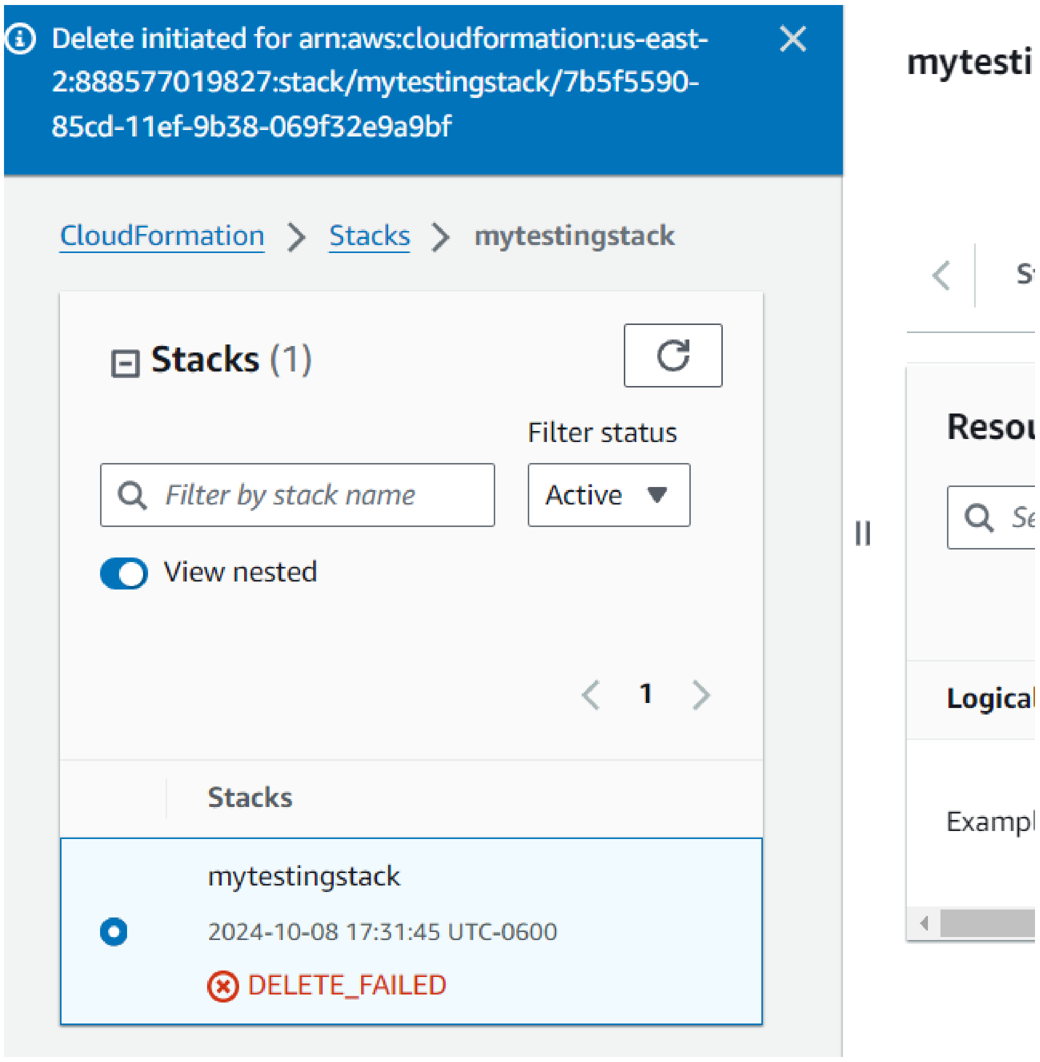
## **Exercise 8.2:** Troubleshoot a Failed Stack Deletion

In this section, we will be deploying a template on AWS CloudFormation. Create a new stack and input a template from a source file. Create a JSON file with the following.

```
{
  "Resources" : {
    "ExampleBucket" : {
      "Type": "AWS::S3::Bucket"
    }
  },
  "Outputs" : {
    "BucketName" : {
      "Value": { "Ref": "ExampleBucket" }
    }
  }
}
```

Upload a few other objects to the template, such as the index and error html files created previously, and maybe some random photos you can find online. Once those are uploaded, delete the stack, and wait until its status is *Failed*

Take note of the error when the stack when it finaly reached DELETE_FAILED as a state.


Delete initiated for arn:aws:cloudformation:us-east-2:888577019827:stack/mytestingstack/7b5f5590-85cd-11ef-9b38-069f32e9a9bf

CloudFormation > Stacks > mytestingstack

Stacks (1)

Filter status: Active

Filter by stack name

View nested

| Stacks |
|---|
| mytestingstack<br>2024-10-08 17:31:45 UTC-0600<br>DELETE_FAILED |

Once you reach this point, go ahead and delete the objects in your bucket, then try to delete the stack again.

It should be fully deleted, and only the stack will delete, don't forget to delete the testingbucket created with the stack to clear up resources.

## **Exercise 8.3**: Monitor Stack Update Activity

In this section, we will be reviewing Monitor Stack Update Activity

Deploy this AWS CloudFormation code template, which provisions an Amazon S3 bucket in *your account. This is also the last JSON created from 8.2, you can use that again.*

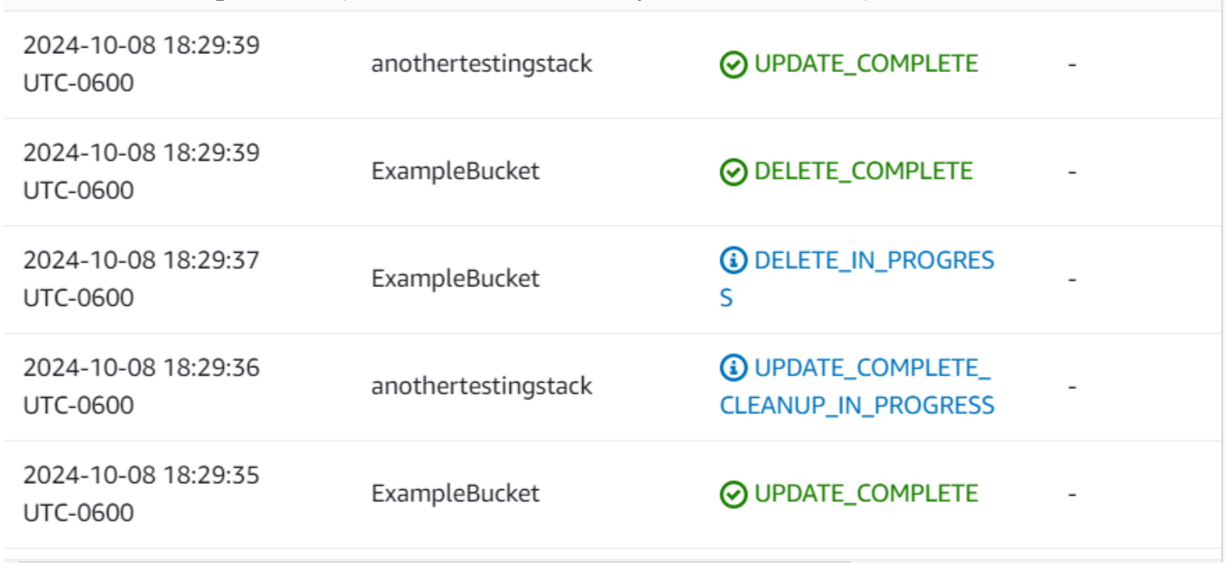
```
{
   "Resources" : {
      "ExampleBucket" : {
         "Type": "AWS::S3::Bucket"
      }
   },
  "Outputs" : {
      "BucketName" : {
         "Value": { "Ref": "ExampleBucket" }
      }
   }
}
```

Now, we are going to upload another template, grab the bucketname that you just created, and fill it in in the below code.

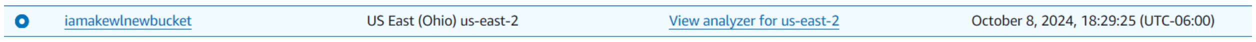
```
{
   "Resources" : {
      "ExampleBucket" : {
         "Type": "AWS::S3::Bucket",
         "Properties": {
            "BucketName": "iamakewlnewbucket"
         }
      }
   },
  "Outputs" : {
      "BucketName" : {
         "Value": { "Ref": "ExampleBucket" }
      }
   }
}
```

Note, this is also a JSON file. Once you have the code filled in, head over to the stack, and at the top right of the panel, there is update. Go into update and choose a different existing template, and upload the new template just made.

Below is a picture of the status history. You ccan see the stack update complete, and deletes the old bucket and updated it (with the same name if you don't touch it) and creates the new bucket *



| | | | |
|---|---|---|---|
| 2024-10-08 18:29:39 UTC-0600 | anothertestingstack | UPDATE_COMPLETE | - |
| 2024-10-08 18:29:39 UTC-0600 | ExampleBucket | DELETE_COMPLETE | - |
| 2024-10-08 18:29:37 UTC-0600 | ExampleBucket | DELETE_IN_PROGRESS | - |
| 2024-10-08 18:29:36 UTC-0600 | anothertestingstack | UPDATE_COMPLETE_CLEANUP_IN_PROGRESS | - |
| 2024-10-08 18:29:35 UTC-0600 | ExampleBucket | UPDATE_COMPLETE | - |

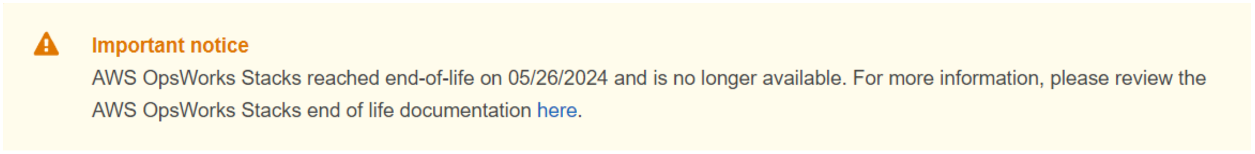


| | | | | |
|---|---|---|---|---|
| ● | iamakewlnewbucket | US East (Ohio) us-east-2 | View analyzer for us-east-2 | October 8, 2024, 18:29:25 (UTC-06:00) |

"iamakewlnewbucket" is the new bucket created from the new template json. Found in AWS S3.

# Chapter 9: Configuration As Code

Unfortunately, the AWS OpsWorks has reached the End of its life cycle as of 10/18/2024. This was put into effect on 5/26/2024.



**Important notice**
AWS OpsWorks Stacks reached end-of-life on 05/26/2024 and is no longer available. For more information, please review the AWS OpsWorks Stacks end of life documentation here.

Since this chapter is focused on OpsWorks, the original coding exercises will be provided from the book "*AWS Certified Developer Official Study Guide: By Nick Alteen, Jennifer Fisher, and Casey Gerena"*

A Solution will be provided following the initial exercises, for the case of migration and understanding the set up for the new way to do this as suggested by AWS.

## Exercise 9.1: Launch a Sample AWS OpsWorks Stacks Environment

1. Launch the **AWS Management Console**.
2. Select **Services** ➢ **AWS OpsWorks**.
3. Select **Add Stack** and select **Sample stack**.
4. Select your preferred operating system (**Linux** or **Windows**).
5. Select **Add Instance** and monitor the stack's progress until it enters the online state. This deploys the app to the stack.
6. Copy the public IP Address, and paste it into a web browser to display the sample app.
7. Open the instance in the **AWS OpsWorks Stacks** console and view the log entries.
8. Verify that the Chef run was a success, and which resources deploy to the instance in the log entries.
9. Update the recipes of the automatically created layer.
10. Remove the deploy recipe.
11. Add a new instance to the stack and monitor its progress.
12. Once the instance is in the online state, view the run logs to verify that the sample website is not deployed to the instance.

## Exercise 9.2: Launch an Amazon ECS Cluster and Containers

1. Launch the Amazon ECS console.
2. Create a new cluster with an Amazon EC2 container instance.
3. Create a new task definition that launches a WordPress and MySQL container.

4. Use the official images from **Docker Hub**:
    1. https://registry.hub.docker.com/wordpress/
    2. https://registry.hub.docker.com/mysql/
5. Create a new service that launches this task definition on the cluster.
6. Copy the public IP address, and paste it into a web browser to access WordPress on the cluster instance.
7. Modify the service to launch two tasks.
8. As the second task attempts to launch, note that this will fail because of the ports configured in the task definition already being registered with the running containers.
9. Launch an additional cluster instance in your cluster.
10. Monitor the service status to verify that the second task deploys to the new cluster instance.

## Exercise 9.3: Migrate an Amazon RDS Database

1. Launch the **Amazon RDS console**.
2. Create a new database instance.
3. Connect to your database and create a user with a password. For example, to create a user with full privileges on MySQL, use the following command:

```
GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' IDENTIFIED BY 'password';
```

4. Launch the **AWS OpsWorks console**.
5. Create two stacks (one “A” stack and one “B” stack). For ease of use, try the sample Linux stack with a Node.js app.
6. Register the RDS database instance that you created with stack A, providing the database username and password you created.
7. Edit the stack’s app to include **Amazon RDS** as a data source. Select the database you registered and provide the database name.
8. Verify that you can connect to your database by creating a simple recipe to output the credentials. Specifically, try to output to the database field of the deploy attributes.
9. Run this recipe to verify that the connection information passes to your nodes.
10. Pass the same connection information into stack A using custom JSON.
11. **Deregister** the database from stack A and **register** it with stack B.
12. Perform the same tasks to verify that connection details pass to the instances in stack B.
13. Remove the custom JSON from stack A to complete the migration.

## Exercise 9.4: Auto Healing in AWS OpsWorks Stacks

1. Launch the **Amazon SNS console**.
2. Create a new notification topic with your email address as a recipient.
3. Launch the **Amazon CloudWatch console**.
4. Create an Amazon CloudWatch Rule.
    1. Edit the JSON version of the rule pattern to use:
    2. {
    3.  "source": [ "aws.opsworks" ],
    4.  "detail": {
    5.   "initiated_by": [
    6.    "auto-healing"
    7.   ]
    8.  }

}

5. Add the Amazon SNS topic that you created as a target.
6. Add permissions to the Amazon SNS topic so that it can be invoked by Amazon CloudWatch Events. An example policy statement is shown here. Replace the value of the Resource block with your topic Amazon Resource Name (ARN).
7. {
8.  "Version": "2008-10-17",
9.  "Id": "AutoHealingNotificationPolicy",
10.  "Statement": [{
11.   "Effect": "Allow",
12.   "Principal": {
13.    "Service": "events.amazonaws.com"
14.   },
15.   "Action": "sns:Publish",
16.   "Resource": "arn:aws:sns:REGION:ACCOUNT:MyTopic"
17.  }]

}

18. Create a stack and add an instance. Make sure that Auto Healing is enabled on the stack.
19. Launch the instance.
20. SSH or RDP into the instance.
21. Uninstall the **AWS OpsWorks Stacks Agent**.
22. Wait until the instance is stopped and started by AWS OpsWorks Stacks. You will receive a notification shortly after this occurs.

**Alternative Route, using AWS Systems Manager Environment.**

MANAGEMENT TOOLS

AWS Systems Manager

Gain Operational Insight and Take Action on AWS Resources.

Get Started with Systems Manager

View operational data for groups of resources, so you can quickly identify and act on any issues that might impact applications that use those resources.

Instead of using amazon OpsWorks, we will now be managing a CloudFormation code stack, inside of AWS Systems Manager, so the CloudFormation will be associated with its EC2 to cloud formation.

## Alternate Exercise 9.1: Launch a Sample AWS Systems Manager Environment

1. **Launch AWS Systems Manager Console**: Log into the AWS Management Console, and navigate to **Systems Manager** under the Services tab.
2. **Create a CloudFormation Stack**: Instead of using OpsWorks stacks, create a CloudFormation stack to define and provision your infrastructure as code.
    - In Systems Manager, use **State Manager** and **Automation** to deploy apps automatically.
3. **Deploy an Instance**:
    - Use **State Manager** to associate an Amazon EC2 instance to your CloudFormation stack and manage configurations.
    - Monitor the deployment progress by checking the instance's state in **EC2 Instances** and **Systems Manager State Manager**.

4. **View Logs**: View the instance logs using **AWS CloudWatch** or **AWS Systems Manager**.
    - Use **Run Command** to execute a command or script to check the instance's success status.
5. **Modify Configuration**: Update your stack's automation document to change the recipes or tasks.
6. **Verify Website Deployment**: Copy the public IP address from EC2 instances, and verify that the web app is deployed.

## Alternate Exercise 9.2: Launch an Amazon ECS Cluster and Containers in Systems Manager

1. **Launch Amazon ECS Console**: Open the **Amazon ECS** console.
2. **Create ECS Cluster**: Launch a new ECS cluster and create EC2 container instances.
3. **Task Definition for WordPress and MySQL**: Use **Docker Hub** official images to define your tasks.
    - Use the official WordPress and MySQL Docker images:
        - WordPress: https://registry.hub.docker.com/wordpress/
        - MySQL: https://registry.hub.docker.com/mysql/
4. **Monitor Deployment with Systems Manager**: Use **Run Command** in Systems Manager to monitor container status and deploy multiple instances if required.
5. **Add a Second Task**: Add a second task to the same service, modifying your service configuration in the ECS console.

## Alternate Exercise 9.3: Migrate an Amazon RDS Database Using AWS Systems Manager

1. **Launch RDS Console**: Create a new Amazon RDS instance from the RDS console.
2. **Connect to Database**: Using **AWS Systems Manager Session Manager**, securely connect to your RDS instance and run MySQL commands to set up users and permissions.
3. **Use AWS Systems Manager Parameter Store**: Store database credentials in **Parameter Store** for secure access by your applications.
4. **Migrate the App to Use Systems Manager**:
    - Use **Automation** in Systems Manager to create a document that migrates the app from one stack to another.
    - Store custom configurations, including the database connection, in **Parameter Store** or **Secrets Manager**.

5. **Modify App Configurations**: Update the app stack to fetch the RDS details using Systems Manager's GetParameter functionality, ensuring connection details are passed correctly.

## Alternate Exercise 9.4: Configure Auto-Healing Event Notifications in AWS Systems Manager

1. **Amazon SNS Notification Setup**: Create an SNS topic as before for receiving notifications.
2. **Create Amazon CloudWatch Rule**: Set up a CloudWatch rule to monitor Systems Manager events instead of OpsWorks.
    - Use event pattern source: [ "aws.ssm" ] to track automated instance healing processes from Systems Manager.
3. **Configure Auto Healing**: Use **State Manager** in Systems Manager to configure automatic healing by associating desired configurations with EC2 instances.
4. **Monitor and Uninstall Agent**: SSH or RDP into the instance using **Session Manager**, uninstall the Systems Manager agent, and observe auto-healing events via CloudWatch and SNS notifications.

These exercises show the functionality of AWS OpsWorks using AWS Systems Manager, leveraging CloudFormation, ECS, RDS, and Systems Manager tools like State Manager, Automation, and Parameter Store for better scalability and control

# Chapter 10: AWS Active Directory

## Exercise 10.1: Setting Up a Simple Active Directory

In this exercise, you will set up an AWS Simple Active Directory (Simple AD). Simple AD is a standalone directory that is powered by a Samba 4 Active Directory Compatible Server.

**Step 1: Create a Virtual Private Cloud**

In this step, you will use the Amazon Virtual Private Cloud (Amazon VPC) wizard in the Amazon VPC console to create a virtual private cloud. The wizard steps create a VPC with a /16 IPv4 CIDR block and attach an internet gateway to the VPC.

1. In the AWS Management Console navigation pane, choose **VPC** and then choose **Launch VPC Wizard**.
2. On the left, select **VPC with a Single Public Subnet**, and then choose **Select**.

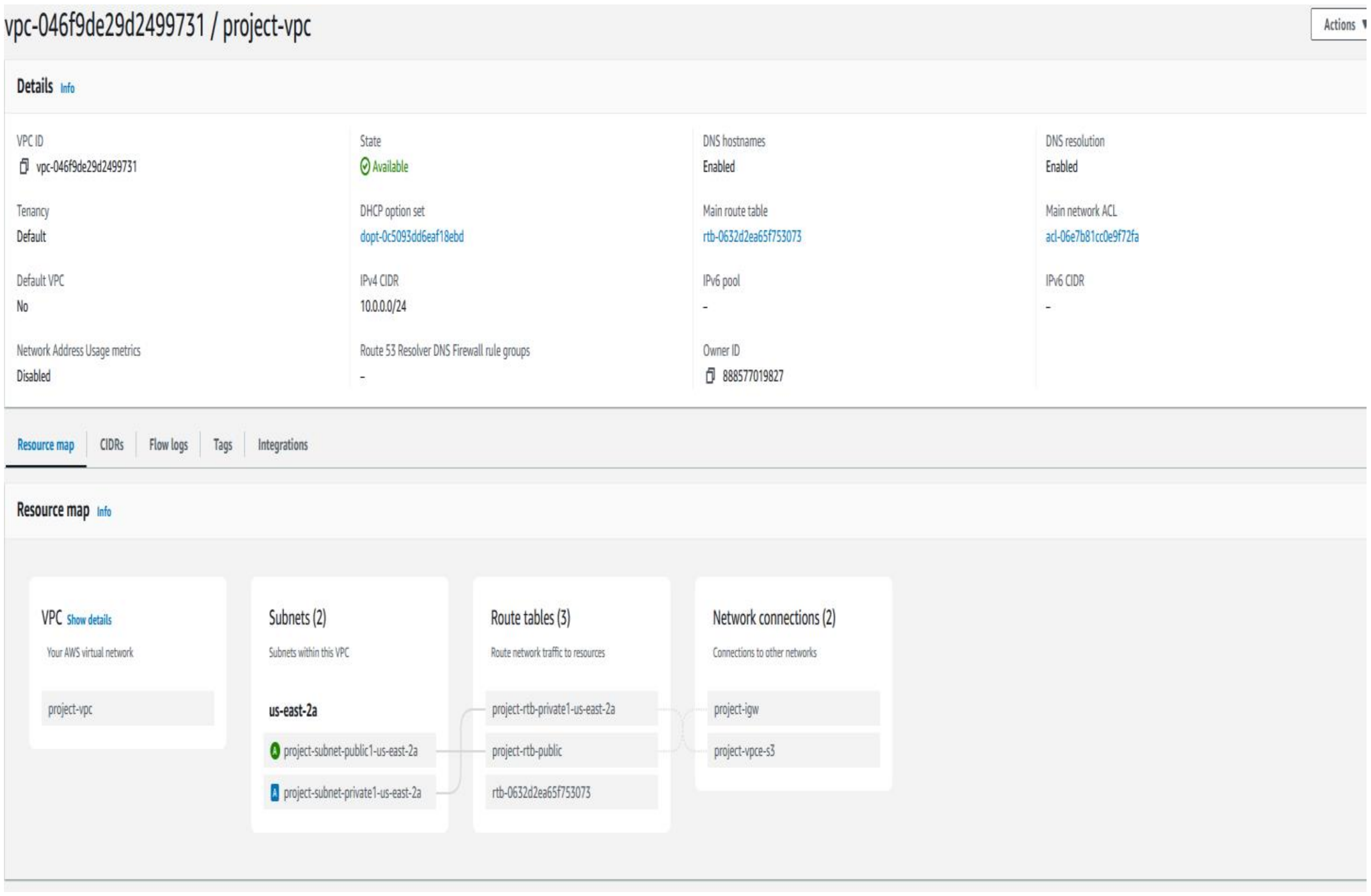
- To communicate with an Active Directory outside of AWS, you must create the Simple AD directory in a public subnet.

3. On the next page, enter the following settings:
    1. Enter a valid, unused IP CIDER block (for example **10.40.0.0/16**).
    2. Choose a valid name (for example, **simple-ad-demo**).
    3. Choose a valid subnet (for example, **10.40.1.0/24**)
    4. Choose an Availability Zone in which to create the subnet. (Record your selection; you will need this information for the next step.)
4. Choose **Create VPC**.

- You have launched a VPC that has a public subnet, an internet gateway attached to it, and the necessary route table and security group configurations to allow traffic to flow between the subnet and the internet gateway. However, because Simple AD is a highly available service, you must create a second public subnet on this VPC.

5. Navigate to the VPC dashboard and choose **Subnets**.
6. Choose **Create Subnet**.
7. On the next page, enter the following settings:
    1. Choose a name tag.
    2. Choose the VPC that you created in the previous steps.
    3. Choose an Availability Zone that is different from the one selected in the previous steps.
    4. Choose a valid subnet.

8. Choose **Create VPC**.
9. You should now be on the following VPC Dashboard



vpc-046f9de29d2499731 / project-vpc

Details Info

| | | | |
|---|---|---|---|
| VPC ID<br>vpc-046f9de29d2499731 | State<br>Available | DNS hostnames<br>Enabled | DNS resolution<br>Enabled |
| Tenancy<br>Default | DHCP option set<br>dopt-0c5093dd6eaf18ebd | Main route table<br>rtb-0632d2ea65f753073 | Main network ACL<br>acl-06e7b81cc0e9f72fa |
| Default VPC<br>No | IPv4 CIDR<br>10.0.0.0/24 | IPv6 pool<br>- | IPv6 CIDR<br>- |
| Network Address Usage metrics<br>Disabled | Route 53 Resolver DNS Firewall rule groups<br>- | Owner ID<br>888577019827 | |

We will now create the actual Managed Active Directory, since simple is no longer available.

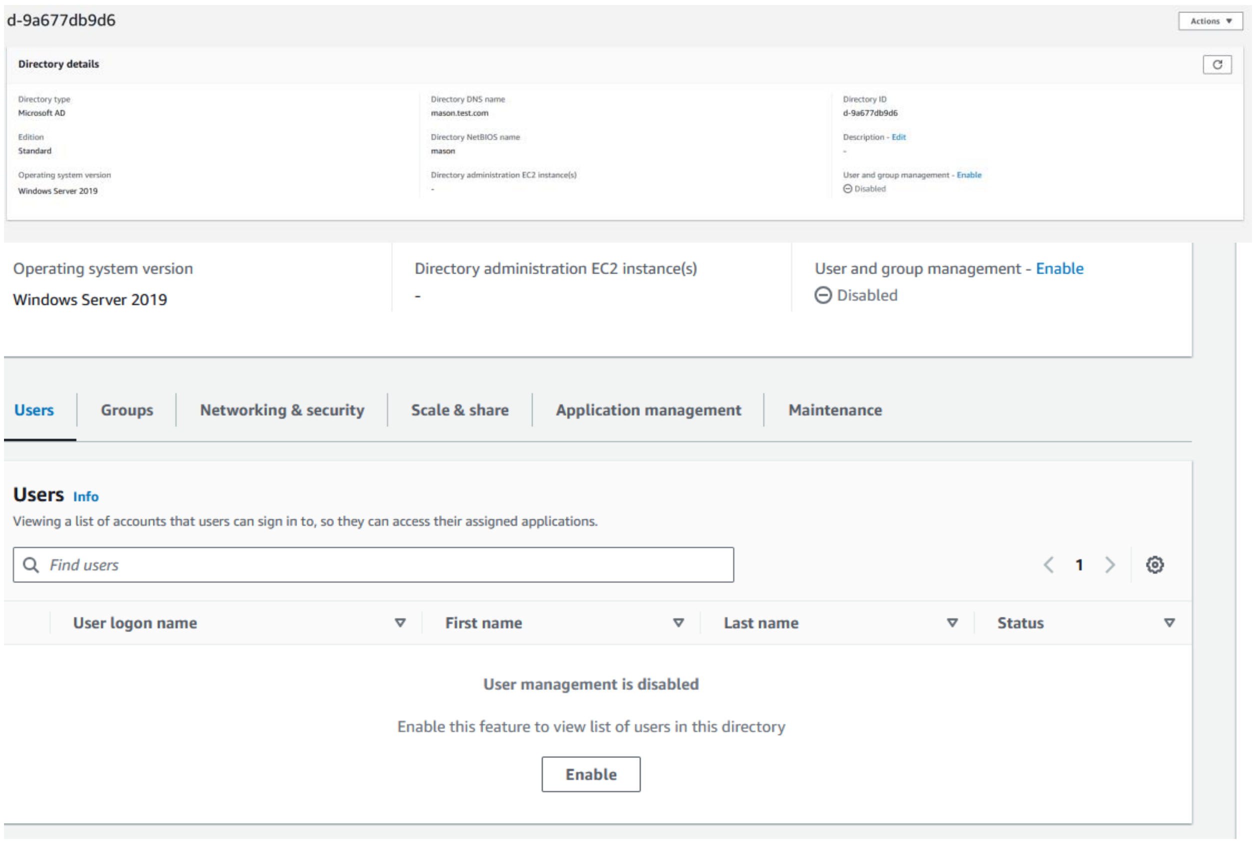
Create your AWS Managed Microsoft AD directory using the AWS Management Console.

1. In the AWS Directory Service console navigation pane, choose **Directories** and then choose **Set up directory**.
2. On the **Select directory type** page, select **AWS Managed AD** and then choose **Next**.
3. On the **Enter directory information** page, provide the following and then choose Next.
    1. Directory size (Small or Large).
    2. Directory DNS name.
    3. (Optional) Directory NetBIOS name (CORP, for example). If one is not provided, a name is created by default.
    4. An administrator password, which must be 8–24 characters in length. It must also contain at least one character from three of the following four categories: uppercase letters, lowercase letters, numbers, or nonalphanumeric characters.

    5. (Optional) Description of the directory. This is useful in tracking your services within AWS.

4. On the **VPC and subnets** page, provide the following information, and then choose **Next**.
    1. For **VPC**, choose the VPC you created earlier (simple-ad-demo).
    2. Under **Subnets**, choose the two subnets you created for the domain controllers.
5. Review your settings and choose **Create directory**.

- It takes 5–10 minutes to create your directory. You may need to refresh the page. When the directory creation is complete, the Status value changes to *Active*.


d-9a677db9d6

**Directory details**

| | | |
|---|---|---|
| Directory type: Microsoft AD | Directory DNS name: mason.test.com | Directory ID: d-9a677db9d6 |
| Edition: Standard | Directory NetBIOS name: mason | Description - Edit: - |
| Operating system version: Windows Server 2019 | Directory administration EC2 instance(s): - | User and group management - Enable: Disabled |

| | | |
|---|---|---|
| Operating system version: Windows Server 2019 | Directory administration EC2 instance(s): - | User and group management - Enable: Disabled |

**Users** Info

Viewing a list of accounts that users can sign in to, so they can access their assigned applications.

| User logon name | First name | Last name | Status |
|---|---|---|---|

User management is disabled

Enable this feature to view list of users in this directory

## Exercise 10.3: Amazon Cloud Directory.

In this exercise, you will set up an Amazon Cloud Directory.

**Step 1: Create a Schema**

You'll first create a schema (which defines objects in a directory) and then assign that schema to a directory. A single schema can be assigned to multiple directories, and a directory can have multiple schemas assigned to it (though typically it does not).

1. In the AWS Services console navigation pane, under Security, Identity & Compliance, choose **Directory Service ➢ Schemas**.
2. To create a custom schema based on an existing one, in the table listing the schemas, select the schema named **person**.
3. Choose **Actions**.
4. Choose **Download schema**.
5. In the location where you downloaded the schema, rename the file to **test-person**.
6. On the **Schemas** page, choose **Upload new schema**.
7. Select test-person and choose **Upload**.
8. To prevent modifications to the schema, choose **schema test-person**.
9. Choose **Actions**.
10. In **Major Version**, enter the identifier **1**, and choose **Publish**.

- You are returned to the Schemas page. You have two versions of the test-person schema: one schema version shows versions and is listed under State as Published; the other schema version does not show versions and is listed under State as Development.
- You have successfully created a schema that you will use to create a directory.

**Step 2: Create a Directory**

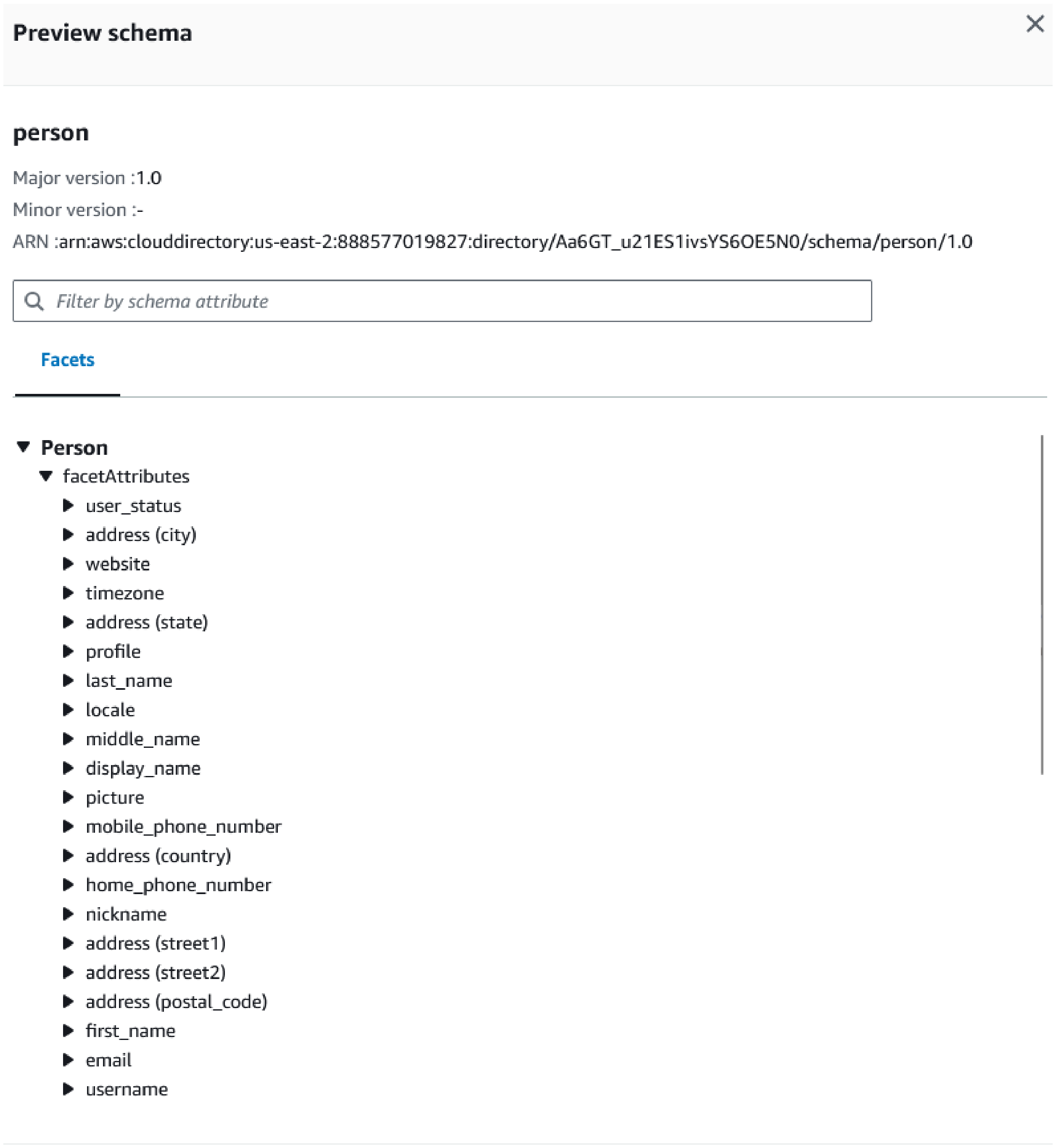
Before you can create a directory in Cloud Directory, Directory Service requires that you first apply a schema to it. A directory cannot be created without a schema and typically has one schema applied to it.

Create a directory that uses the schema you created in step 1.

1. In the AWS Directory Service console navigation pane, under Security, Identity & Compliance, choose **Directory Service ➢ Directories**.
2. Choose **Set up Cloud Directory**.
3. Under **Choose a schema to apply to your new directory**, in **Cloud Directory name**, enter **test-cloud-directory**.
4. Choose **Personal or Custom schema**.
5. Select the custom schema named test-person with the Status of Published, and then choose **Next**.
6. Review the directory information and make the necessary changes. When the information is correct, choose **Create**.

You have successfully created a Cloud Directory. You can modify and delete the directory, including the schema associated with the directory.

Here is the finished schema, of personal directory inside of the cloud directory.


Preview schema

person

Major version :1.0
Minor version :-
ARN :arn:aws:clouddirectory:us-east-2:888577019827:directory/Aa6GT_u21ES1ivsYS6OE5N0/schema/person/1.0

Filter by schema attribute

Facets

- Person
  - facetAttributes
    - user_status
    - address (city)
    - website
    - timezone
    - address (state)
    - profile
    - last_name
    - locale
    - middle_name
    - display_name
    - picture
    - mobile_phone_number
    - address (country)
    - home_phone_number
    - nickname
    - address (street1)
    - address (street2)
    - address (postal_code)
    - first_name
    - email
    - username

## Exercise 10.4: Amazon Cognito

1. In this exercise, you will set up Amazon Cognito, which is the service that provides authentication, authorization, and user management for web and mobile applications. In the AWS Directory Services console navigation pane, under Security, Identity & Compliance, choose **Cognito** and then choose **Manage User Pools**.

2. Provide a name for your user pool. Enter **admin-group**.
3. Specify how a user signs in. In this example, select user name, and then choose **Next step**.
4. Retain the default settings and choose **Next step**.

- You can set MFA as optional or required. After a user pool is configured, you cannot change the MFA setting. Amazon Cognito uses Amazon SNS to send SMS messages. If MFA is enabled, you must assign a role with the correct policy to send SMS messages.

5. Retain the default settings and choose **Next step**.
6. On the **Attributes** page, retain the default settings for email customization, and choose **Next step**.
7. To manage the AWS infrastructure, apply tags. Enter the following information, and then choose **Next step**:
    1. In **Tag Key**, enter **user**
    2. In **Tag Value**, enter **admin-user**.
8. Select **No** and choose **Next step**.

- Amazon Cognito can detect and retain your user's device. This step enables you to configure that capability. In this example, however, you will select **No**. Choose **Next step**.

9. Retain the default settings and choose **Next step**.

- You can configure how client applications gain access to the user pool. In this exercise, no access is granted.

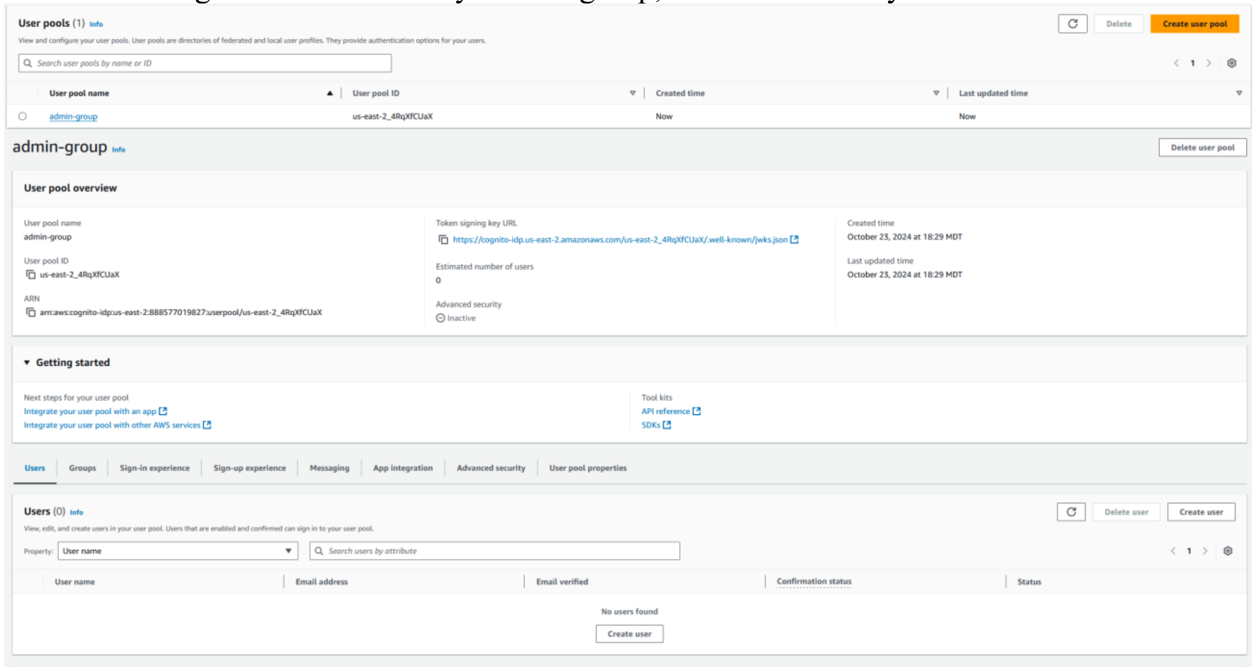
10. Retain the default settings and choose **Next step**.

- You can configure AWS Lambda functions that can be triggered during the Amazon Cognito operation. For this exercise, you will not configure any Lambda functions.

11. On the **Review** page, review your configurations, and choose **Create pool**.

You have successfully created a user pool in Amazon Cognito.

Below is a image of the successfully created group, if follow correctly.



**User pools** (1) Info

View and configure your user pools. User pools are directories of federated and local user profiles. They provide authentication options for your users.

Search user pools by name or ID

| User pool name | User pool ID | Created time | Last updated time |
|---|---|---|---|
| admin-group | us-east-2_4RqXfCUaX | Now | Now |

admin-group Info

**User pool overview**

User pool name
admin-group

User pool ID
us-east-2_4RqXfCUaX

ARN
arn:aws:cognito-idp:us-east-2:888577019827:userpool/us-east-2_4RqXfCUaX

Token signing key URL
https://cognito-idp.us-east-2.amazonaws.com/us-east-2_4RqXfCUaX/.well-known/jwks.json

Estimated number of users
0

Advanced security
Inactive

Created time
October 23, 2024 at 18:29 MDT

Last updated time
October 23, 2024 at 18:29 MDT

**Getting started**

Next steps for your user pool
Integrate your user pool with an app
Integrate your user pool with other AWS services

Tool kits
API reference
SDKs

**Users** (0) Info

View, edit, and create users in your user pool. Users that are enabled and confirmed can sign in to your user pool.

Property: User name

Search users by attribute

| User name | Email address | Email verified | Confirmation status | Status |
|---|---|---|---|---|

No users found

Create user

# Chapter 11: SQS and Data Streams

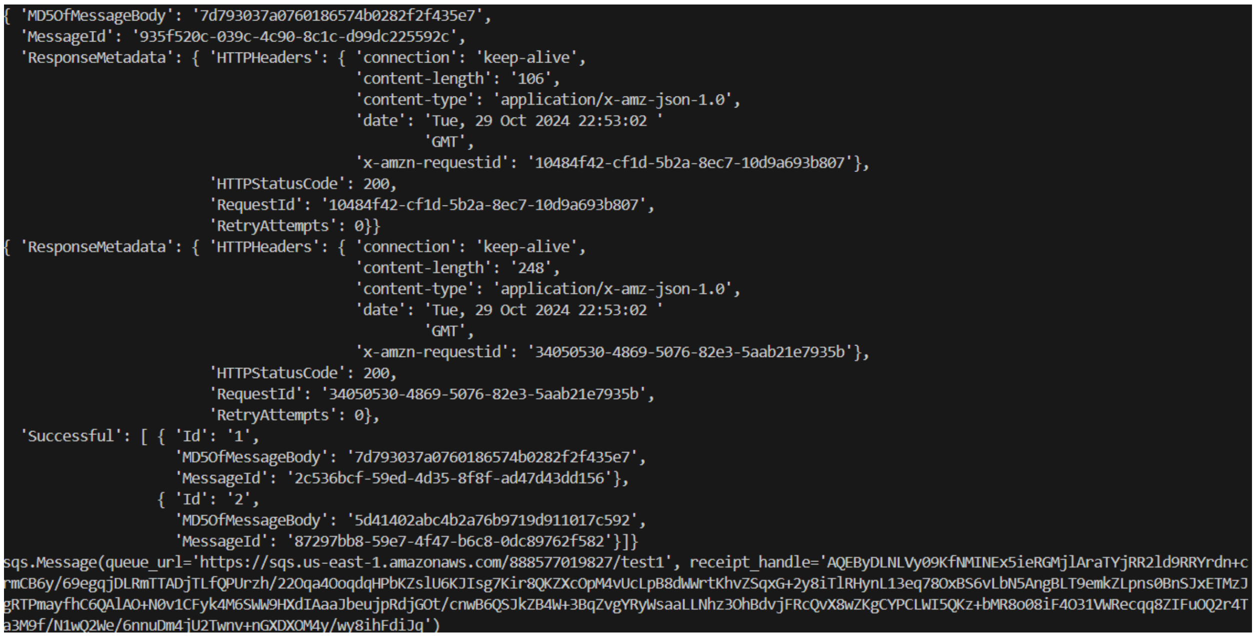
## Exercise 11.1: Create an Amazon SQS Queue

In this exercise, you will use the AWS SDK for Python (Boto) to create an Amazon SQS queue, and then you will put messages in the queue. Finally, you will receive messages from this queue and delete them.

Enter the following code into your development environment for Python or the IPython shell.

```
# Test SQS.
import boto3

# Pretty print.
import pprint
pp = pprint.PrettyPrinter(indent=2)

# Create queue.
sqs = boto3.resource('sqs')
queue = sqs.create_queue(QueueName='test1')
print(queue.url)

# Get existing queue.
queue = sqs.get_queue_by_name(QueueName='test1')
print(queue.url)

# Get all queues.
for queue in sqs.queues.all(): print queue

# Send message.
response = queue.send_message(MessageBody='world')
pp.pprint(response)

# Send batch.
response = queue.send_messages(Entries=[
    { 'Id': '1', 'MessageBody': 'world' },
    { 'Id': '2', 'MessageBody': 'hello' } ])
pp.pprint(response)

# Receive and delete all messages.
for message in queue.receive_messages():
    pp.pprint(message)
    message.delete()

# Delete queue.
queue.delete()
```

Below is the terminal output if the full queue was successfully executed.



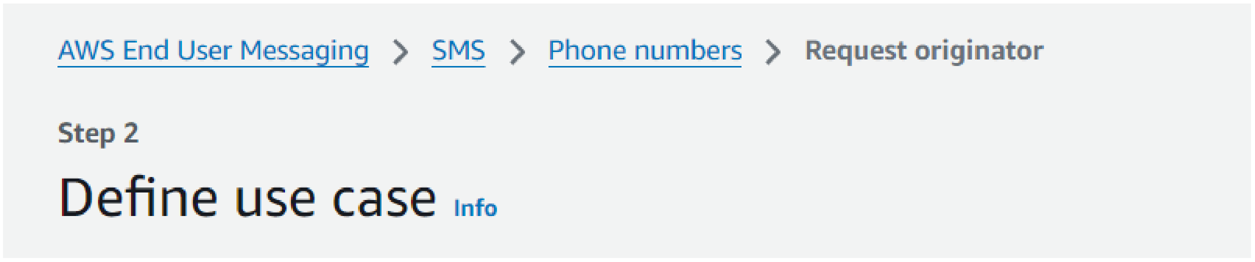
```
{ 'MD5OfMessageBody': '7d793037a0760186574b0282f2f435e7',
  'MessageId': '935f520c-039c-4c90-8c1c-d99dc225592c',
  'ResponseMetadata': { 'HTTPHeaders': { 'connection': 'keep-alive',
                                         'content-length': '106',
                                         'content-type': 'application/x-amz-json-1.0',
                                         'date': 'Tue, 29 Oct 2024 22:53:02 '
                                                 'GMT',
                                         'x-amzn-requestid': '10484f42-cf1d-5b2a-8ec7-10d9a693b807'},
                        'HTTPStatusCode': 200,
                        'RequestId': '10484f42-cf1d-5b2a-8ec7-10d9a693b807',
                        'RetryAttempts': 0}}
{ 'ResponseMetadata': { 'HTTPHeaders': { 'connection': 'keep-alive',
                                         'content-length': '248',
                                         'content-type': 'application/x-amz-json-1.0',
                                         'date': 'Tue, 29 Oct 2024 22:53:02 '
                                                 'GMT',
                                         'x-amzn-requestid': '34050530-4869-5076-82e3-5aab21e7935b'},
                        'HTTPStatusCode': 200,
                        'RequestId': '34050530-4869-5076-82e3-5aab21e7935b',
                        'RetryAttempts': 0},
  'Successful': [ { 'Id': '1',
                    'MD5OfMessageBody': '7d793037a0760186574b0282f2f435e7',
                    'MessageId': '2c536bcf-59ed-4d35-8f8f-ad47d43dd156'},
                  { 'Id': '2',
                    'MD5OfMessageBody': '5d41402abc4b2a76b9719d911017c592',
                    'MessageId': '87297bb8-59e7-4f47-b6c8-0dc89762f582'}]}
sqs.Message(queue_url='https://sqs.us-east-1.amazonaws.com/888577019827/test1', receipt_handle='AQEByDLNLVy09KfNMINEx5ieRGMjlAraTYjRR2ld9RRYrdn+c
rmCB6y/69egqjDLRmTTADjTLfQPUrzh/22Oqa4OoqdqHPbKZslU6KJIsg7Kir8QKZXcOpM4vUcLpB8dWWrtKhvZSqxG+2y8iTlRHynL13eq78OxBS6vLbN5AngBLT9emkZLpns0BnSJxETMzJ
gRTPmayfhC6QAlAO+N0v1CFyk4M6SWW9HXdIAaaJbeujpRdjGOt/cnwB6QSJkZB4W+3BqZvgYRyWsaaLLNhz3OhBdvjFRcQvX8wZKgCYPCLWI5QKz+bMR8o08iF4O31VWRecqq8ZIFuOQ2r4T
a3M9f/N1wQ2We/6nnuDm4jU2Twnv+nGXDXOM4y/wy8ihFdiJq')
```

## Exercise 11.2: Send a text message to your phone

In this exercise, you will use Amazon SNS to publish an SMS message to your mobile phone. This solution can be useful when you run a job that will take several hours to complete, and you do not want to wait for it to finish. Instead, you can have your app send you an SMS text message when it is done. Since this no longer applies, this is how you would send it if the client you are working with is still using AWS SNS

```
import boto3


# Create SNS client.

sns_client = boto3.client('sns')


# Send message to your mobile number.

# (Replace dummy mobile number with your number.)

sns_client.publish(

  PhoneNumber='1-222-333-3333',

  Message='Hello from your app')
```

Now, since SNS has moved to AWS End User Messaging, we will go through that instead. Go to SMS>Phone Numbers> Request Originator



AWS End User Messaging > SMS > Phone numbers > Request originator

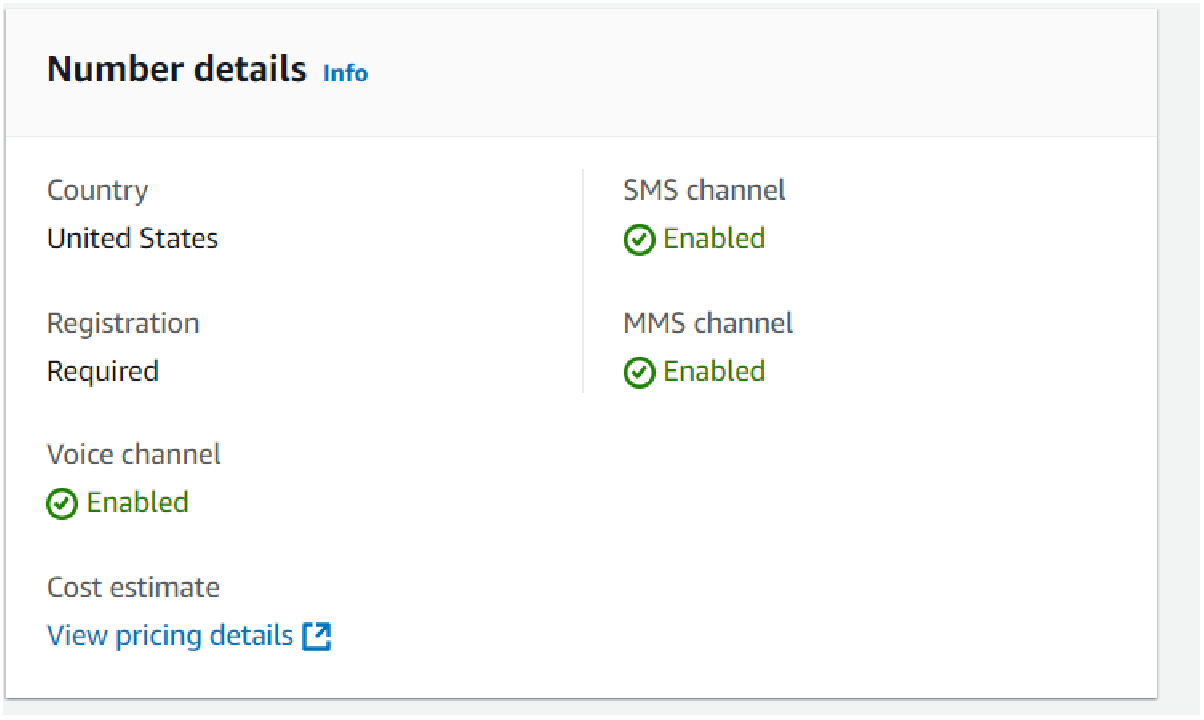
Step 2

Define use case Info

You would then define the amount of messages you plan to send, or you can go to next step.

For SMS messages, you pay per message, where emails only cost about $1 per 10,000 emails.

You final setup for the number should like this, then when you are ready you can publish SMS messages from the SMS dashboard.



Number details Info

Country
United States

Registration
Required

Voice channel
✓ Enabled

Cost estimate
View pricing details

SMS channel
✓ Enabled

MMS channel
✓ Enabled

## 11.3: Create an Amazon Kinesis Data Stream and Write/Read Data

In this exercise, you will create an Amazon Kinesis data stream, put records on it (write to the stream), and then get those records back (read from the stream). At the end, you will delete the stream.

import boto3

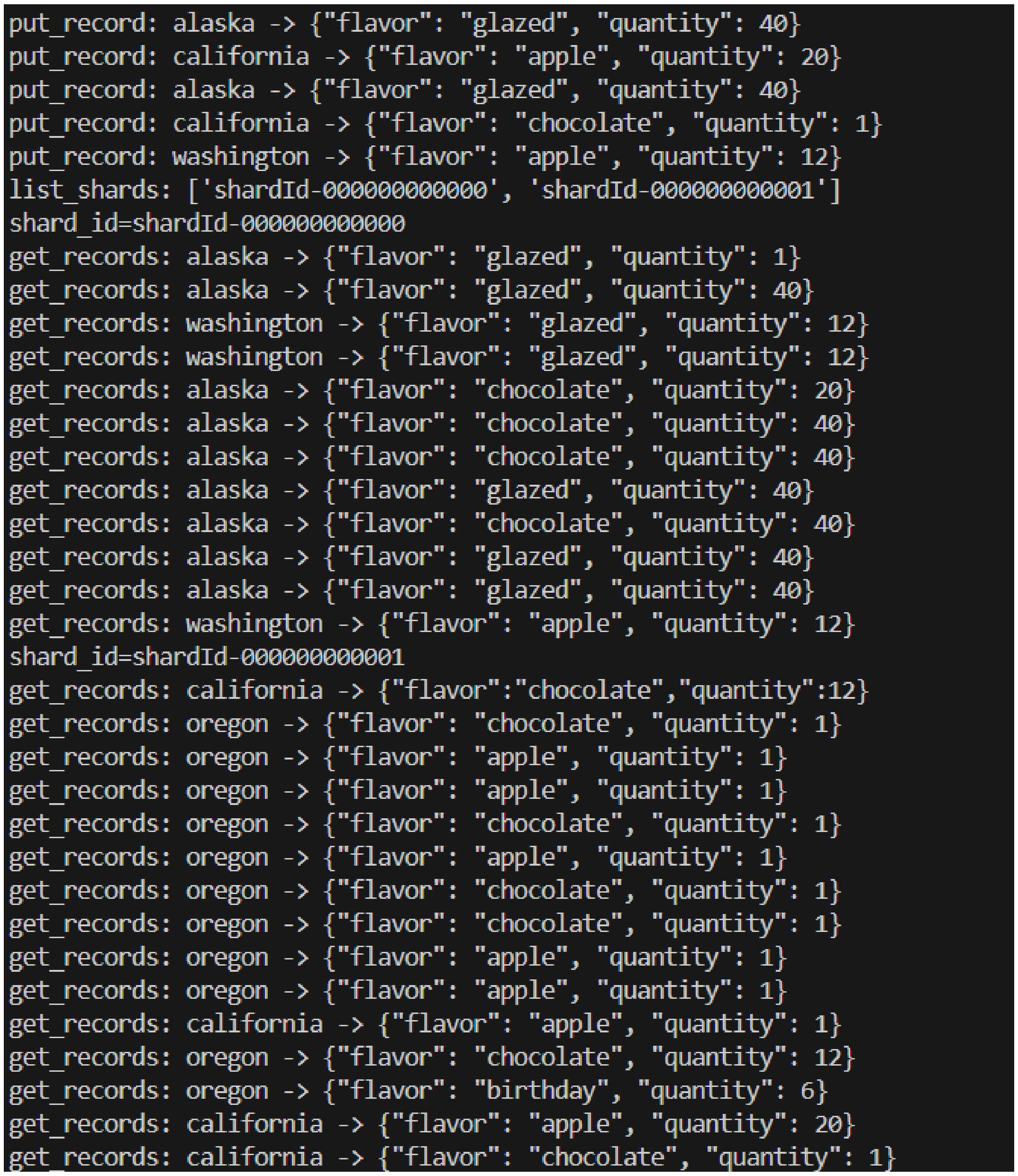
import random

```
import json

# Create the client.
kinesis_client = boto3.client('kinesis')

# Create the stream.
kinesis_client.create_stream(
  StreamName='donut-sales',
  ShardCount=2)

# Wait for stream to be created.
waiter = kinesis_client.get_waiter('stream_exists')
waiter.wait(StreamName='donut-sales')

# Store each donut sale using location as partition key.
location = 'california'
data = b'{"flavor":"chocolate","quantity":12}'
kinesis_client.put_record(
    StreamName='donut-sales',
    PartitionKey=location, Data=data)
print("put_record: " + location + " -> " + data)

# Next lets put some random records.

# List of location, flavors, quantities.
locations = ['california', 'oregon', 'washington', 'alaska']
flavors = ['chocolate', 'glazed', 'apple', 'birthday']
quantities = [1, 6, 12, 20, 40]
```

```
# Generate some random records.
for i in xrange(20):

    # Generate random record.
    flavor = random.choice(flavors)
    location = random.choice(locations)
    quantity = random.choice(quantities)
    data = json.dumps({"flavor": flavor, "quantity": quantity})

    # Put record onto the stream.
    kinesis_client.put_record(
        StreamName='donut-sales',
        PartitionKey=location, Data=data)
    print("put_record: " + location + " -> " + data)

# Get the records.

# Get shard_ids.
response = kinesis_client.list_shards(StreamName='donut-sales')
shard_ids = [shard['ShardId'] for shard in response['Shards']]
print("list_shards: " + str(shard_ids))

# For each shard_id print out the records.
for shard_id in shard_ids:

    # Print current shard_id.
    print("shard_id=" + shard_id)
    # Get a shard iterator from this shard.
    # TRIM_HORIZON means start from earliest record.
    response = kinesis_client.get_shard_iterator(
        StreamName='donut-sales',
        ShardId=shard_id,
        ShardIteratorType='TRIM_HORIZON')
    shard_iterator = response['ShardIterator']

    # Get records on shard and print them out.
    response = kinesis_client.get_records(ShardIterator=shard_iterator)
    records = response['Records']
    for record in records:
        location = record['PartitionKey']
        data = record['Data']
        print("get_records: " + location + " -> " + data)

# Delete the stream.
kinesis_client.delete_stream(
  StreamName='donut-sales')

# Wait for stream to be deleted.
waiter = kinesis_client.get_waiter('stream_not_exists')
waiter.wait(StreamName='donut-sales')
```

Observe the output and how all the records for a specific location occur in the same shard. This is because they have the same partition keys. All records with the same partition key are sent to the same shard.

Below is a snippet terminal output that you should be observing.



```
put_record: alaska -> {"flavor": "glazed", "quantity": 40}
put_record: california -> {"flavor": "apple", "quantity": 20}
put_record: alaska -> {"flavor": "glazed", "quantity": 40}
put_record: california -> {"flavor": "chocolate", "quantity": 1}
put_record: washington -> {"flavor": "apple", "quantity": 12}
list_shards: ['shardId-000000000000', 'shardId-000000000001']
shard_id=shardId-000000000000
get_records: alaska -> {"flavor": "glazed", "quantity": 1}
get_records: alaska -> {"flavor": "glazed", "quantity": 40}
get_records: washington -> {"flavor": "glazed", "quantity": 12}
get_records: washington -> {"flavor": "glazed", "quantity": 12}
get_records: alaska -> {"flavor": "chocolate", "quantity": 20}
get_records: alaska -> {"flavor": "chocolate", "quantity": 40}
get_records: alaska -> {"flavor": "chocolate", "quantity": 40}
get_records: alaska -> {"flavor": "glazed", "quantity": 40}
get_records: alaska -> {"flavor": "chocolate", "quantity": 40}
get_records: alaska -> {"flavor": "glazed", "quantity": 40}
get_records: alaska -> {"flavor": "glazed", "quantity": 40}
get_records: washington -> {"flavor": "apple", "quantity": 12}
shard_id=shardId-000000000001
get_records: california -> {"flavor":"chocolate","quantity":12}
get_records: oregon -> {"flavor": "chocolate", "quantity": 1}
get_records: oregon -> {"flavor": "apple", "quantity": 1}
get_records: oregon -> {"flavor": "apple", "quantity": 1}
get_records: oregon -> {"flavor": "chocolate", "quantity": 1}
get_records: oregon -> {"flavor": "apple", "quantity": 1}
get_records: oregon -> {"flavor": "chocolate", "quantity": 1}
get_records: oregon -> {"flavor": "chocolate", "quantity": 1}
get_records: oregon -> {"flavor": "apple", "quantity": 1}
get_records: oregon -> {"flavor": "apple", "quantity": 1}
get_records: california -> {"flavor": "apple", "quantity": 1}
get_records: oregon -> {"flavor": "chocolate", "quantity": 12}
get_records: oregon -> {"flavor": "birthday", "quantity": 6}
get_records: california -> {"flavor": "apple", "quantity": 20}
get_records: california -> {"flavor": "chocolate", "quantity": 1}
```

## Exercise 11.4: Step function state machine 1.

In this exercise, you will create an AWS Step Functions state machine. The state machine will extract price and quantity from the input and inject the billing amount into the output.

This state machine will calculate how much to bill a customer based on the price and quantity of an item they purchased.

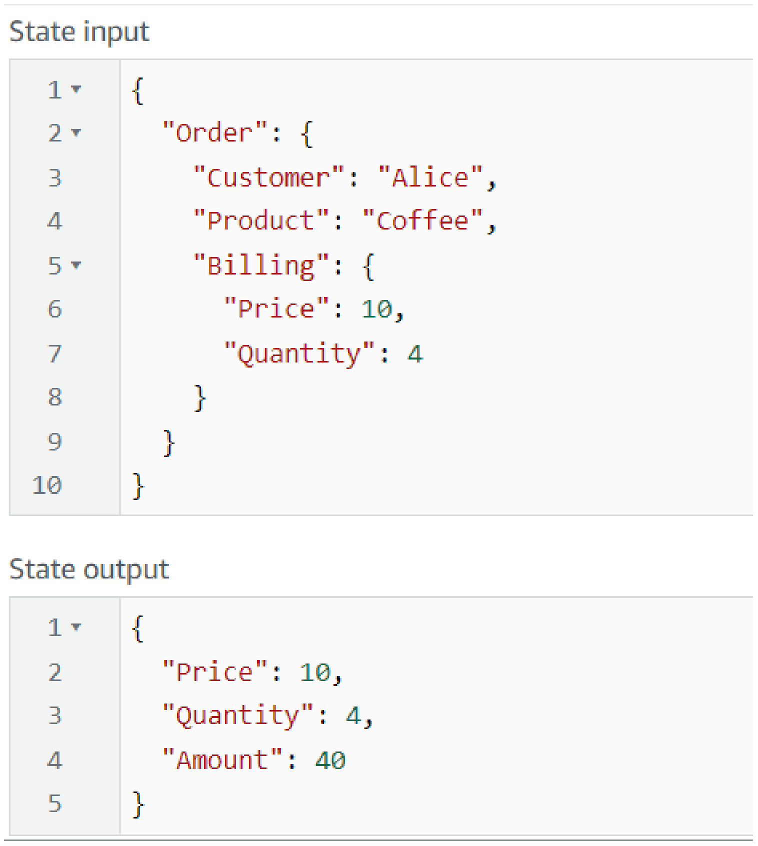
1. Sign in to the AWS Management Console and open the Step Functions console at https://console.aws.amazon.com/step-functions/.
2. Select **Get Started**.

3. On the **Define state machine** page, select **Custom**.
4. In **Name type**, enter **order-machine**.
5. Enter the code for the state machine definition under the code section (You need to move off of design).

This is the code that you downloaded at the beginning of the exercises.

```
{
 "StartAt": "CreateOrder",
 "States": {
  "CreateOrder": {
   "Type": "Pass",
   "Result": {
    "Order" :  {
     "Customer" : "Alice",
     "Product" : "Coffee",
     "Billing" : { "Price": 10.0, "Quantity": 4.0 }
    }
   },
   "Next": "CalculateAmount"
  },
  "CalculateAmount": {
   "Type": "Pass",
   "Result": 40.0,
   "ResultPath": "$.Order.Billing.Amount",
   "OutputPath": "$.Order.Billing",
   "End": true
  }
 }
}
```

Below is an output of the machine, showing its working.


State input

```
{
  "Order": {
    "Customer": "Alice",
    "Product": "Coffee",
    "Billing": {
      "Price": 10,
      "Quantity": 4
    }
  }
}
```

State output

```
{
  "Price": 10,
  "Quantity": 4,
  "Amount": 40
}
```

## Exercise 11.5: Create an AWS Step Functions State Machine 2

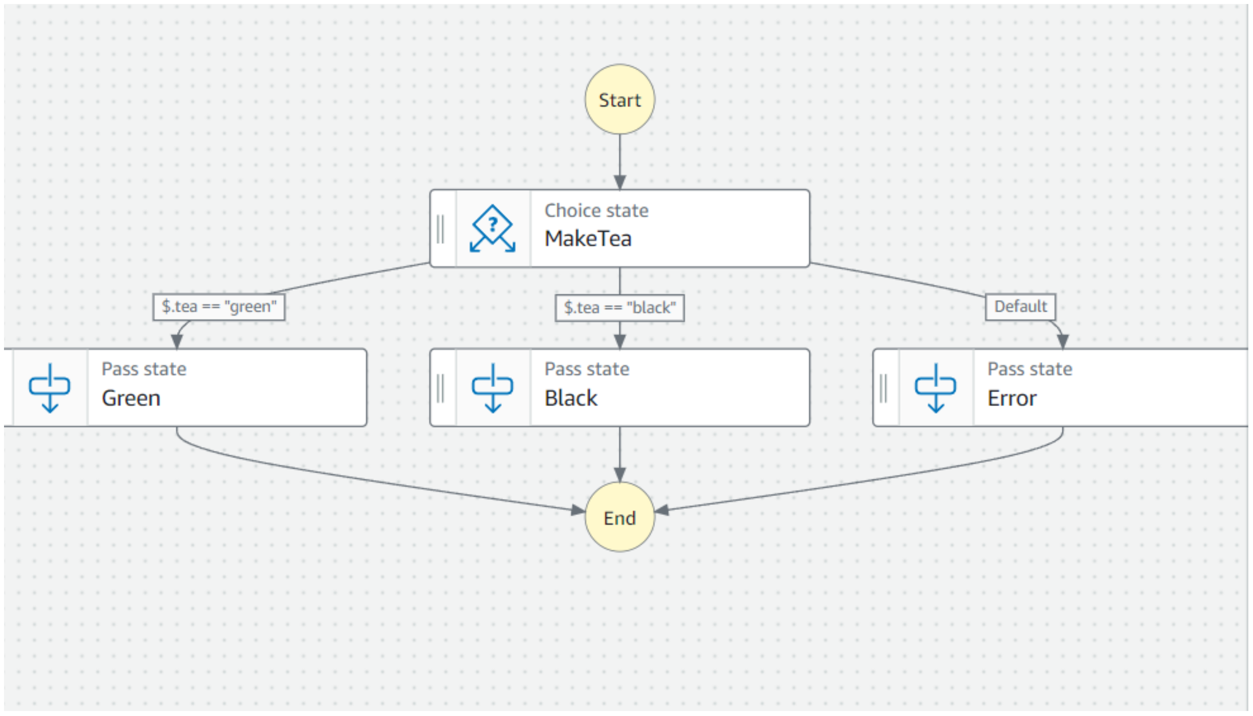
In this exercise, you will create an AWS Step Functions state machine. The state machine will contain a conditional branch. It will use the Choice state to choose which state to transition to next.

The state machine inspects the input and based on it decides whether the user ordered green tea, ordered black tea, or entered invalid input.

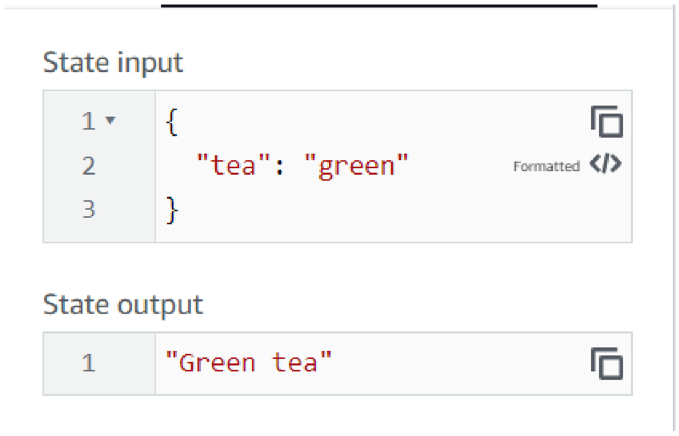
Get to the same place before with a new machine, and insert the following for this machines code definition.

This Relationship Diagram the machine makes in AWS helps us understand exactly what the code above is doing.

```
{
  "Comment": "Input should look like {'tea':'green'} with double quotes instead of single.",
  "StartAt": "MakeTea",
"States": {
  "MakeTea": {
    "Type": "Choice",
    "Choices": [
      {
        "Variable": "$.tea",
        "StringEquals": "green",
        "Next": "Green"
      },
      {
        "Variable": "$.tea",
        "StringEquals": "black",
        "Next": "Black"
      }
    ],
    "Default": "Error"
  },
  "Green": {
    "Type": "Pass",
    "End": true,
    "Result": "Green tea"
  },
  "Black": {
    "Type": "Pass",
    "End": true,
    "Result": "Black tea"
  },
  "Error": {
      "Type": "Pass",
      "End": true,
      "Result": "Bad input"
    }
  }
}
```



Below are the sample input/output based on what the code can handle.

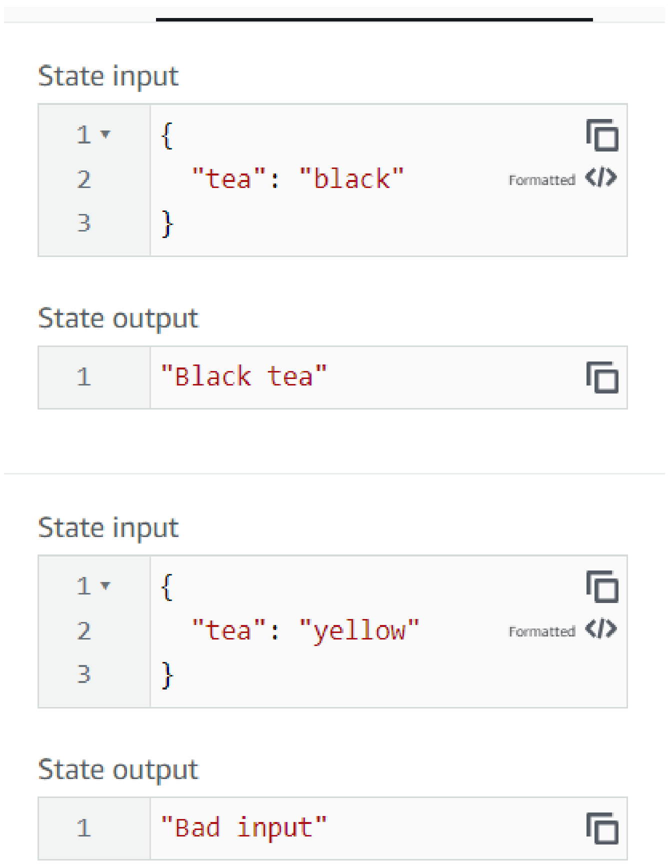


State input

```
{
  "tea": "green"
}
```

State output

```
"Green tea"
```



State input

```
{
  "tea": "black"
}
```

State output

```
"Black tea"
```

State input

```
{
  "tea": "yellow"
}
```

State output

```
"Bad input"
```

# Chatper 12: AWS LAMBDA Functions

## Exercise 12.1: Create an Amazon S3 Bucket for CSV Ingestion

For this, we will create two buckets, one for CSV Ingestion and one as a JSON output. We will also need an AWS Lambda Function to process the CSV.

First, lets create a bucket.

*import boto3*

*# Variables for the bucket name and the region we will be using.*

*# Important Note: s3 Buckets are globally unique, as such you need to change the name of the bucket to something else.*

*# Important Note: If you would like to use us-east-1 as the region, when making the s3.create_bucket call, then do not specify any region.*

*bucketName = "shoe-company-2018-ingestion-csv-demo"*

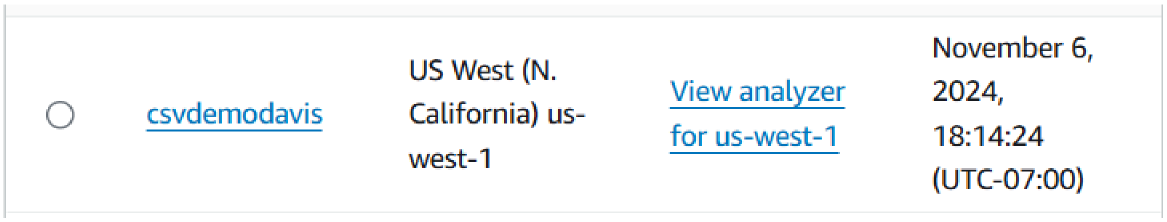
*bucketRegion = "us-west-1"*

*# Creates an s3 Resource; this is a higher level API type service for s3.*

*s3 = boto3.resource('s3')*

*# Creates a bucket*

*bucket = s3.create_bucket(ACL='private',Bucket=bucketName,CreateBucketConfiguration ={'LocationConstraint': bucketRegion})*



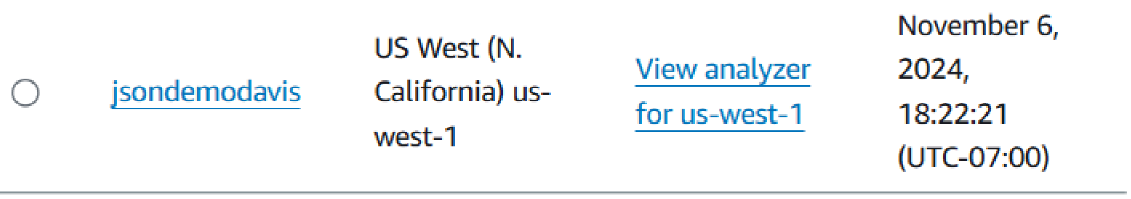
| | | | | |
|---|---|---|---|---|
| ○ | csvdemodavis | US West (N. California) us-west-1 | View analyzer for us-west-1 | November 6, 2024, 18:14:24 (UTC-07:00) |

## Exercise 12.2: Create an Amazon S3 Bucket for Final Output .JSON file.

We will now create the second bucket.

*import boto3*

```
# Variables for the bucket name and the region we will be using.

# Important Note: s3 Buckets are globally unique, as such you need to change the name of the bucket to something else.

   # Important Note: If you would like to use us-east-1 as the region, when making the s3.create_bucket call, then do not specify any region.

bucketName = "shoe-company-2018-final-json-demo"

bucketRegion = "us-west-1"


# Creates an s3 Resource; this is a higher level API type service for s3.

s3 = boto3.resource('s3')


# Creates a bucket

bucket = s3.create_bucket(ACL='private',Bucket=bucketName,CreateBucketConfiguration={'LocationConstraint': bucketRegion})
```



| | | | | |
|---|---|---|---|---|
| ○ | jsondemodavis | US West (N. California) us-west-1 | View analyzer for us-west-1 | November 6, 2024, 18:22:21 (UTC-07:00) |

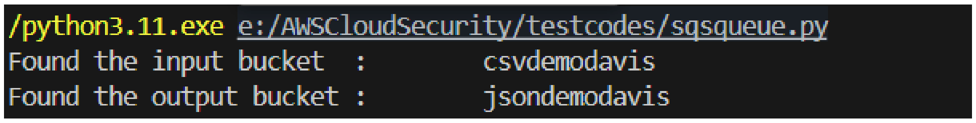
## Exercise 12.3: Verify List Buckets.

*To verify both buckets, use Python 3 and run the following:*

```
import boto3

# Variables for the bucket name and the region we will be using.

# Important Note: Be sure to use the same bucket names you used in the previous two exercises.

bucketInputName = "shoe-company-2018-ingestion-csv-demo"

bucketOutputName = "shoe-company-2018-final-json-demo"

bucketRegion = "us-west-1"


# Creates an s3 Resource; this is a higher level API type service for s3.

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

```
# Get all of the buckets
bucket_iterator = s3.buckets.all()

# Loop through the buckets

for bucket in bucket_iterator:
    if bucket.name == bucketInputName:
        print("Found the input bucket\t:\t", bucket.name)
    if bucket.name == bucketOutputName:
        print("Found the output bucket\t:\t", bucket.name)
```

Your output in whatever IDE you use should verify in some way, in visual studio code, it should look like this.



```
/python3.11.exe e:/AWSCloudSecurity/testcodes/sqsqueue.py
Found the input bucket  :       csvdemodavis
Found the output bucket :       jsondemodavis
```

## Exercise 12.4: Prepare the AWS Lambda Function:

To perform the conversion between the buckets, the AWS CLI and Python SDK work together.

Run the following code.

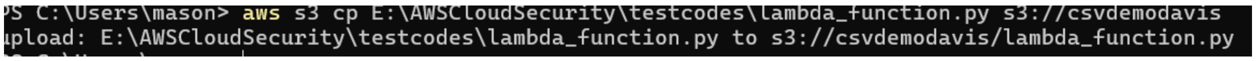
```
import boto3
import csv
import json
import time
# The csv and json modules provide functionality for parsing
# and writing csv/json files. We can use these modules to
# quickly perform a data transformation
# You can read about the csv module here:
```

```
# https://docs.python.org/2/library/csv.html
# and JSON here:
# https://docs.python.org/2/library/json.html

# Create an s3 Resource: https://boto3.readthedocs.io/en/latest/guide/resources.html
s3 = boto3.resource('s3')
csv_local_file = '/tmp/input-payroll-data.csv'
json_local_file = '/tmp/output-payroll-data.json'

# Change this value to whatever you named the output s3 bucket in the previous exercise
output_s3_bucket = 'jsondemodavis'

def lambda_handler(event, context):

    # Need to get the bucket name
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Download the file to our AWS Lambda container environment
    try:
        s3.Bucket(bucket_name).download_file(key, csv_local_file)
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same
region as this function.'.format(key, bucket_name))
        raise e

    # Open the csv and json files
    csv_file = open(csv_local_file, 'r')
    json_file = open(json_local_file, 'w')
    # Get a csv DictReader object to convert file to json
    dict_reader = csv.DictReader( csv_file )

    # Create an Employees array for JSON, use json.dumps to pass in the string
    json_conversion = json.dumps({'Employees': [row for row in dict_reader]})

    # Write to our json file
    json_file.write(json_conversion)

        # Close out the files
    csv_file.close()
    json_file.close()

    # Upload finished file to s3 bucket
    try:
        s3.Bucket(output_s3_bucket).upload_file(json_local_file, 'final-output-payroll.json')
    except Exception as e:
        print(e)
        print('Error uploading object {} to bucket {}. Make sure the file paths are correct.'.format(key,
bucket_name))
        raise e

    print('Payroll processing completed at: ', time.asctime( time.localtime(time.time()) ) )
    return 'Payroll conversion from CSV to JSON complete.'
```

Now with the Lambda set up, we will upload this file to Amazon S3.

```
aws s3 cp lambda_function.zip s3://shoe-company-2018-ingestion-csv-demo
```

Below is the CLI result command after the above line was submitted successfully.



```
S C:\Users\mason> aws s3 cp E:\AWSCloudSecurity\testcodes\lambda_function.py s3://csvdemodavis
pload: E:\AWSCloudSecurity\testcodes\lambda_function.py to s3://csvdemodavis/lambda_function.py
```

## Exercise 12.5: Create AWS IAM Roles.

The IAM roles are needed so our AWS Lambda function has proper permissions to execute the function with AWS CLI.

Create the below .json file.

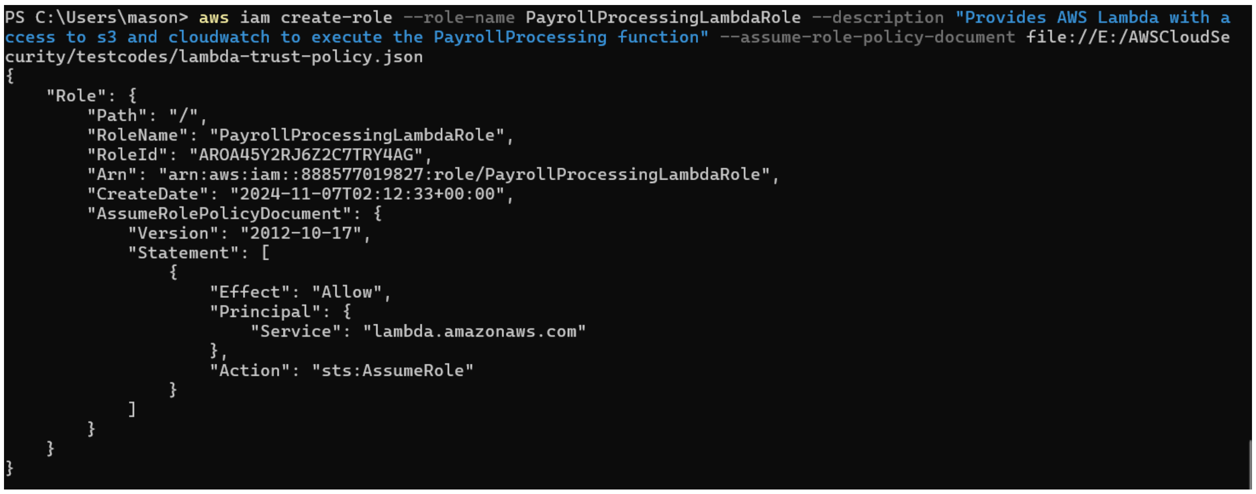
*lambda-trust-policy.json*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Now that the file is created, run the following in the AWS CLI.

```
aws iam create-role -–role-name PayrollProcessingLambdaRole --description "Provides AWS Lambda with access to s3 and cloudwatch to execute the PayrollProcessing function" --assume-role-policy-document file://lambda-trust-policy.json
```

Here is the output in the CLI when finished.

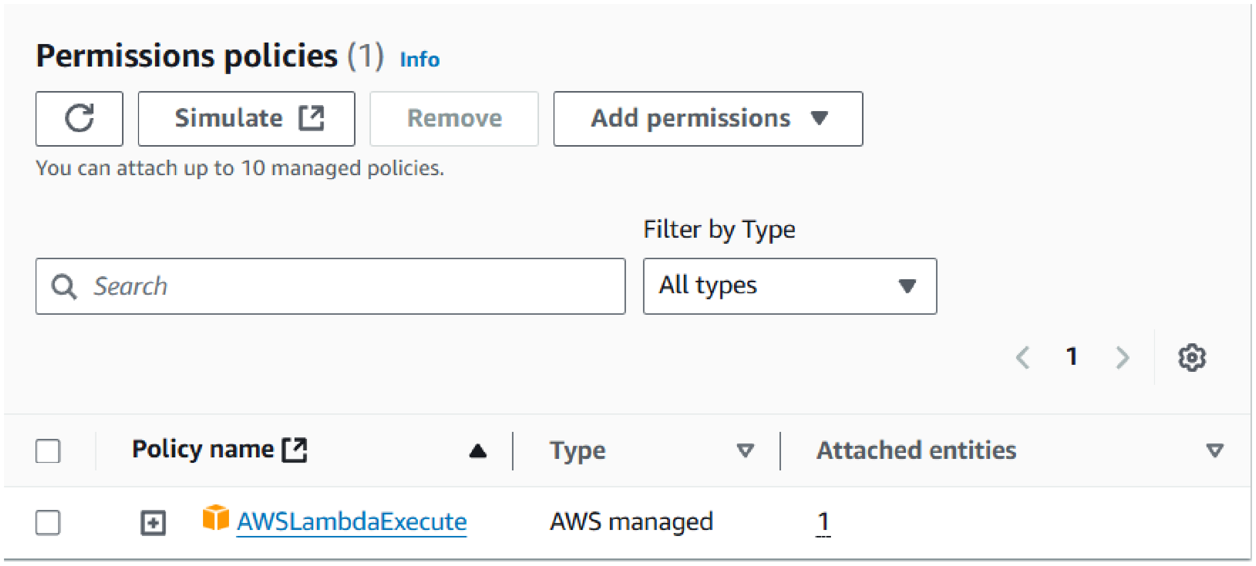


```
PS C:\Users\mason> aws iam create-role --role-name PayrollProcessingLambdaRole --description "Provides AWS Lambda with a
ccess to s3 and cloudwatch to execute the PayrollProcessing function" --assume-role-policy-document file://E:/AWSCloudSe
curity/testcodes/lambda-trust-policy.json
{
    "Role": {
        "Path": "/",
        "RoleName": "PayrollProcessingLambdaRole",
        "RoleId": "AROA45Y2RJ6Z2C7TRY4AG",
        "Arn": "arn:aws:iam::888577019827:role/PayrollProcessingLambdaRole",
        "CreateDate": "2024-11-07T02:12:33+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Now run the following in the CLI to assign a policy to the role.

*aws iam attach-role-policy --role-name PayrollProcessingLambdaRole --policy-arn arn:aws:iam::aws:policy/AWSLambdaExecute*

Now you should just have the one policy attached on your new role.



Permissions policies (1) Info

You can attach up to 10 managed policies.

| Policy name | Type | Attached entities |
|---|---|---|
| AWSLambdaExecute | AWS managed | 1 |

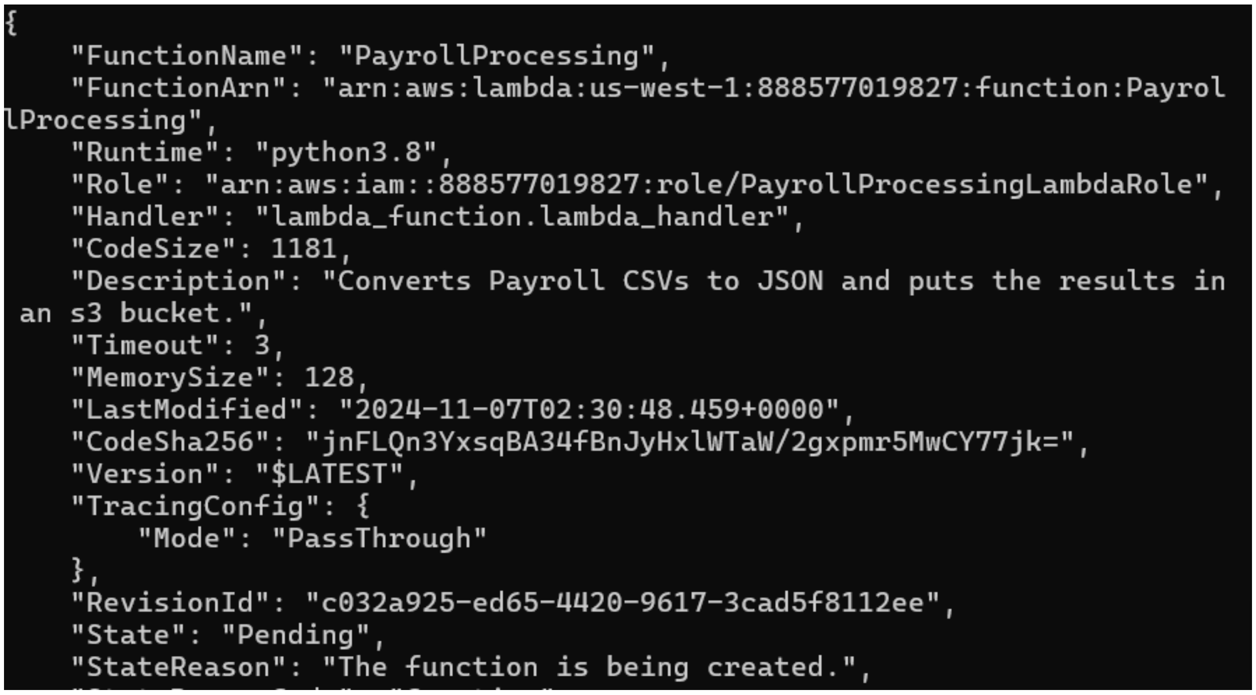
## Exercise 12.6: Create the AWS Lambda Function

Run this AWS CLI Command

*aws lambda create-function --function-name PayrollProcessing --runtime python3.7 --role arn:aws:iam::accountnumber:role/PayrollProcessingLambdaRole --handler lambda_function.lambda_handler --description "Converts Payroll CSVs to JSON and puts the results in an s3 bucket." --timeout 3 --memory-size 128 --code S3Bucket=shoe-company-2018-ingestion-csv-*

*demo,S3Key=lambda_function.zip --tags  Environment="Production",Application="Payroll" --region us-west-1*

If the function is successful, you should receive the following response back.



```
{
    "FunctionName": "PayrollProcessing",
    "FunctionArn": "arn:aws:lambda:us-west-1:888577019827:function:Payrol
lProcessing",
    "Runtime": "python3.8",
    "Role": "arn:aws:iam::888577019827:role/PayrollProcessingLambdaRole",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 1181,
    "Description": "Converts Payroll CSVs to JSON and puts the results in
 an s3 bucket.",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2024-11-07T02:30:48.459+0000",
    "CodeSha256": "jnFLQn3YxsqBA34fBnJyHxlWTaW/2gxpmr5MwCY77jk=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "c032a925-ed65-4420-9617-3cad5f8112ee",
    "State": "Pending",
    "StateReason": "The function is being created.",
```

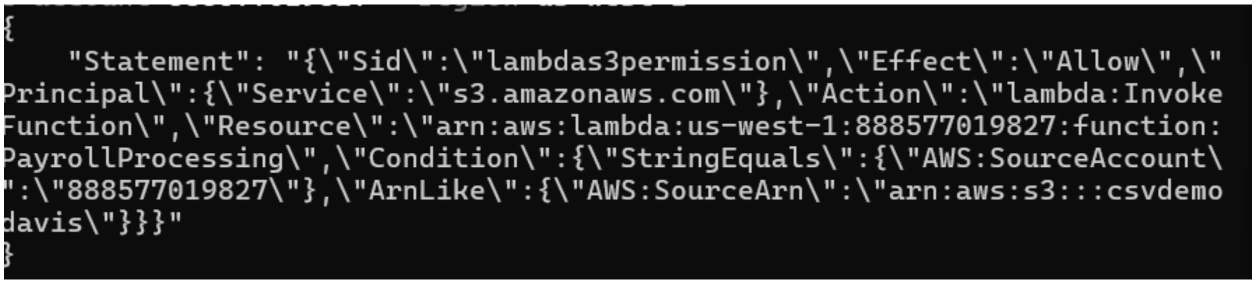
## 12.7: Give Amazon S3 Permission to invoke an AWS Lambda Function

Run this code to the AWS CLI

aws lambda remove-permission --function-name PayrollProcessing --statement-id lambdas3permission --region us-west-1

*aws lambda add-permission --function-name PayrollProcessing --statement-id lambdas3permission --action lambda:InvokeFunction --principal s3.amazonaws.com --source-arn arn:aws:s3:::csvdemodavis --source-account YOURAWSACCOUNTNUMBER --region us-west-1*

This is the successful output



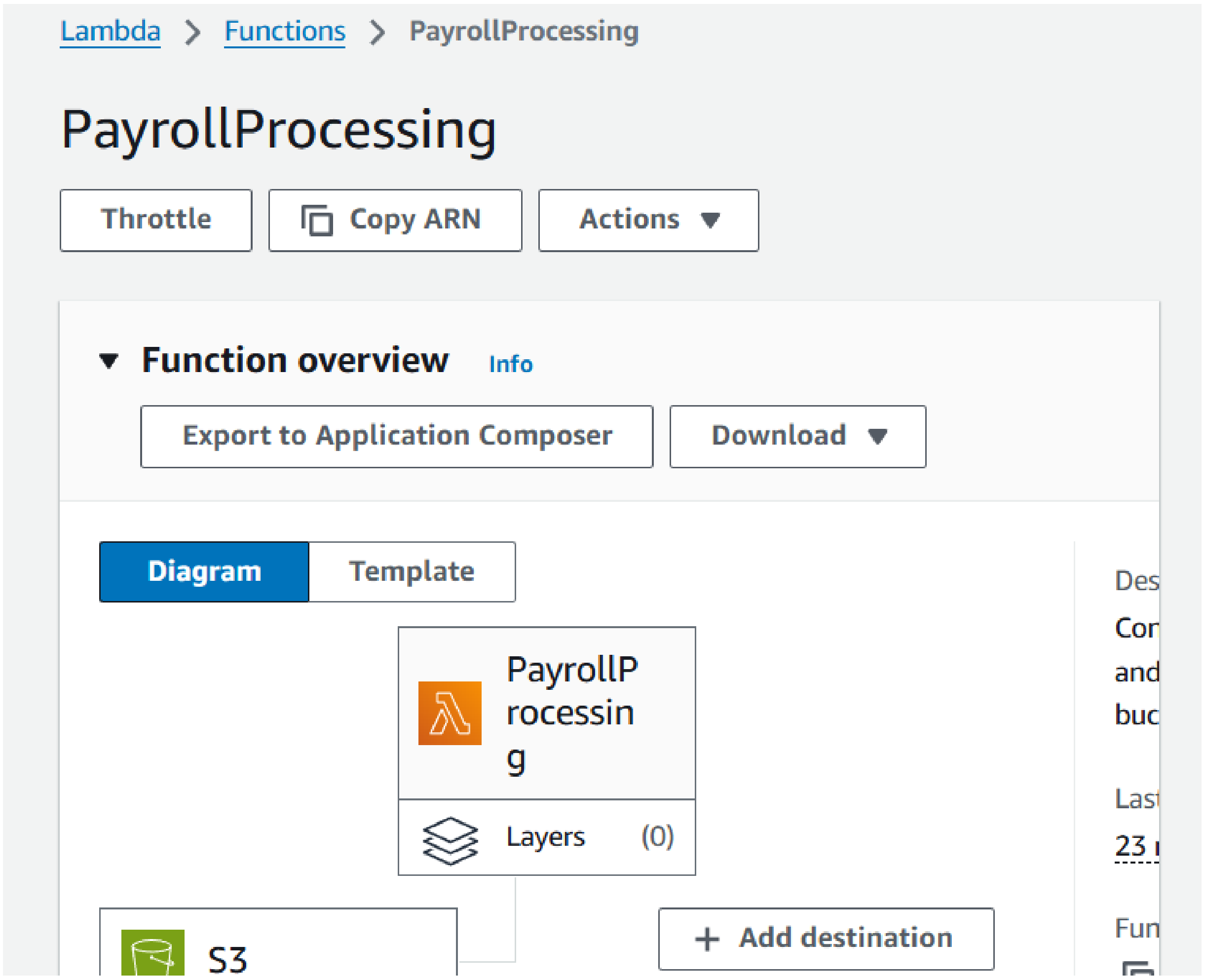
```
{
    "Statement": "{\"Sid\":\"lambdas3permission\",\"Effect\":\"Allow\",\"
Principal\":{\"Service\":\"s3.amazonaws.com\"},\"Action\":\"lambda:Invoke
Function\",\"Resource\":\"arn:aws:lambda:us-west-1:888577019827:function:
PayrollProcessing\",\"Condition\":{\"StringEquals\":{\"AWS:SourceAccount\
":\"888577019827\"},\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:s3:::csvdemo
davis\"}}}"
}
```

## Exercise 12.8: Add the Amazon S3 Event Trigger

In this exercise, add the trigger for s3api commands.

*aws s3api put-bucket-notification-configuration —bucket shoe-company-2018-ingestion-csv-demo —notification-configuration* [*file://notification-config.json*](file://notification-config.json)

It should reply with nothing, to verify, check inside the AWS Lambda Console.



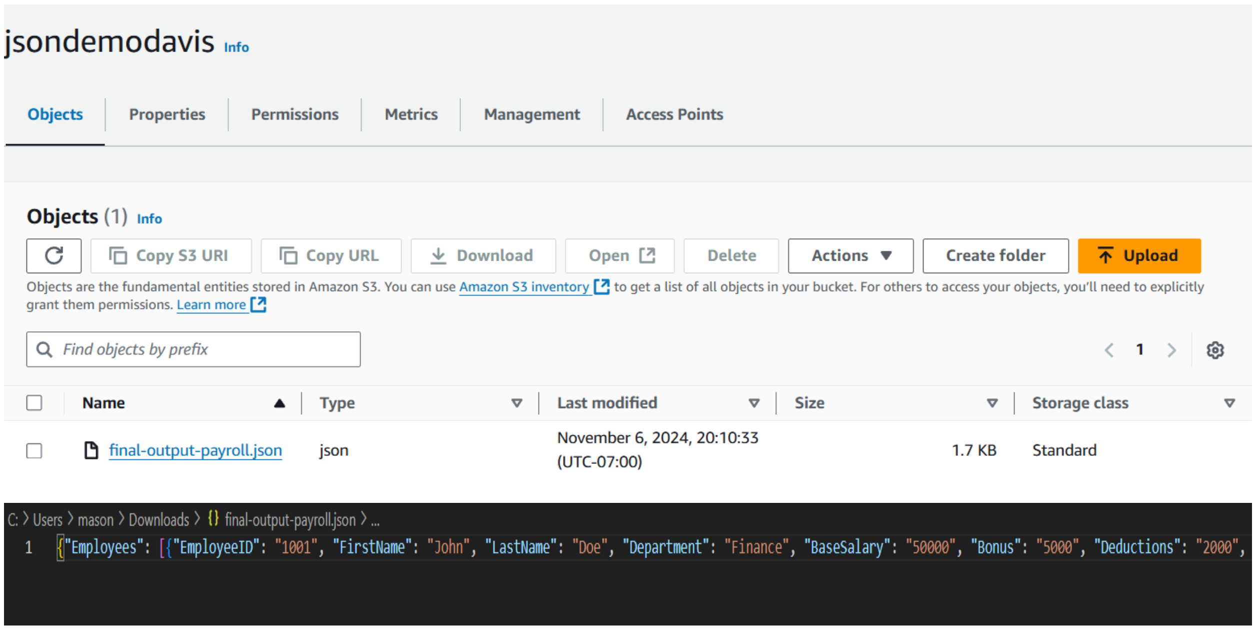
The connected s3 bucket is the trigger we are looking for, but there is more details on it further into the LAMBDA Dashboard.

## Exercise 12.9: Test the AWS Lambda Function

To test the function, use the CLI to upload the csv file to the s3 bucket, then check if the data is transformed in the output .json bucket.

*aws s3 cp input-payroll-data.csv s3://shoe-company-2018-ingestion-csv-demo*

For this, just create a very simple csv file. Now to verify it worked, the outputs below should contains the correct files.



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

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| final-output-payroll.json | json | November 6, 2024, 20:10:33 (UTC-07:00) | 1.7 KB | Standard |

```
C: > Users > mason > Downloads > {} final-output-payroll.json > ...
{"Employees": [{"EmployeeID": "1001", "FirstName": "John", "LastName": "Doe", "Department": "Finance", "BaseSalary": "50000", "Bonus": "5000", "Deductions": "2000",
```

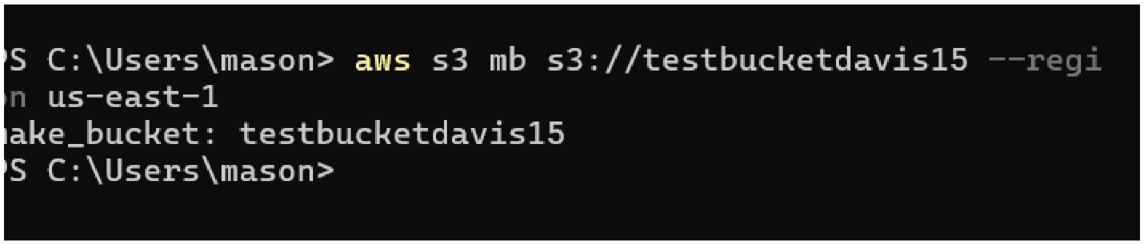
# Chapter 13: Templates and more LAMBDA

## Exercise 13.1: Create an Amazon S3 Bucket for the Swagger Template

This Chapter is much more technical than previously. Be prepared to alter code based on needs, and requirements of what you are setting up. In all cases in this chapter, you will likely need to alter code for future processes.  Extra notes are commented under CMD images.

First, let's create our bucket:

*aws s3 mb s3://my-bucket-name --region us-east-1*



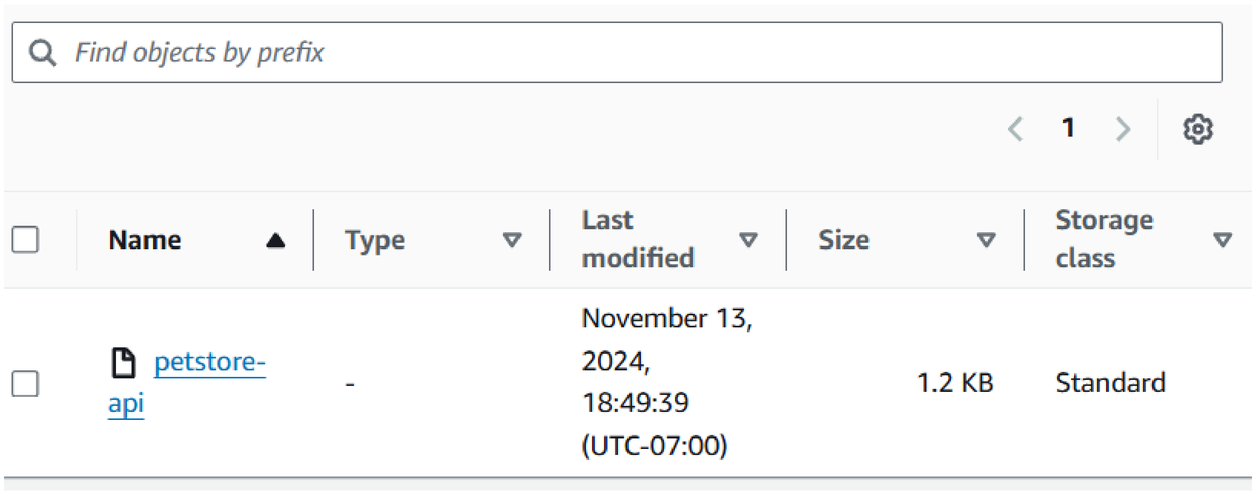
```
'S C:\Users\mason> aws s3 mb s3://testbucketdavis15 --regi
n us-east-1
ake_bucket: testbucketdavis15
'S C:\Users\mason>
```

"*make_bucket: yourbucketname"* is the successful creation of the bucket, so your name wasn't taken, great!

Now lets upload the swagger template from an API request:

*aws s3 cp petstore-api-swagger.yaml s3://my-bucket-name/petstore-api-swagger .yaml*

Remember to change the bucket you your bucket name. If its successful, in the bucket in cloud, you should see the file and nothing else.



| | Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|---|
| ☐ | petstore-api | - | November 13, 2024, 18:49:39 (UTC-07:00) | 1.2 KB | Standard |

Now, use AWS SAM to deploy the serverless infrastructure. To package the SAM, run the following:

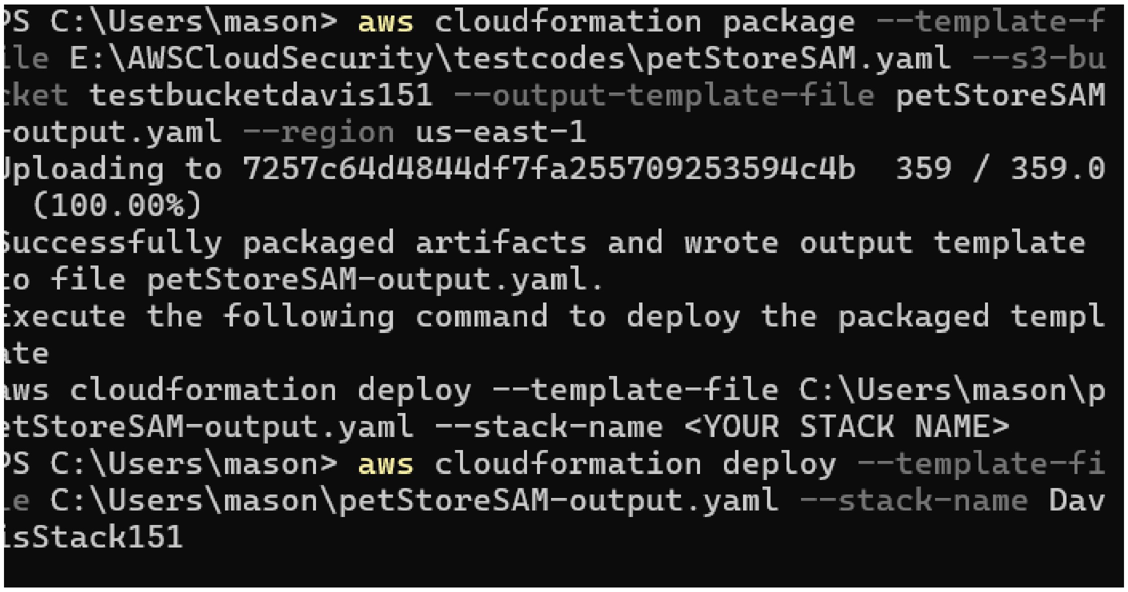
*aws cloudformation package --template-file ./petStoreSAM.yaml *

*--s3-bucket testbucketdavis151 *

*--output-template-file petStoreSAM-output.yaml *

*--region us-east-1*

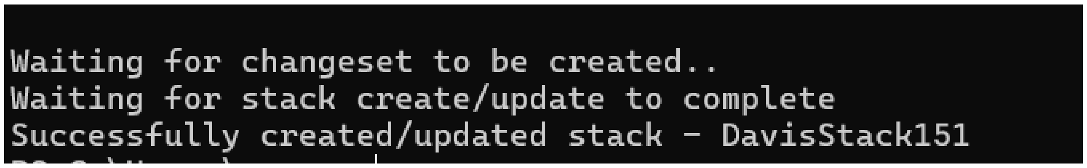
If that was successful, command prompts will tell you what to put in next to deploy the stack.



```
PS C:\Users\mason> aws cloudformation package --template-f
ile E:\AWSCloudSecurity\testcodes\petStoreSAM.yaml --s3-bu
cket testbucketdavis151 --output-template-file petStoreSAM
-output.yaml --region us-east-1
Uploading to 7257c64d4844df7fa255709253594c4b  359 / 359.0
 (100.00%)
Successfully packaged artifacts and wrote output template
to file petStoreSAM-output.yaml.
Execute the following command to deploy the packaged templ
ate
aws cloudformation deploy --template-file C:\Users\mason\p
etStoreSAM-output.yaml --stack-name <YOUR STACK NAME>
PS C:\Users\mason> aws cloudformation deploy --template-fi
le C:\Users\mason\petStoreSAM-output.yaml --stack-name Dav
isStack151
```

You may need to grant the Stack IAM access with –capabilities CAPABILITY_IAM after the stack name.

This is the successful creation prompt.



```
Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - DavisStack151
```

Now let’s get the results from the stack

*aws cloudformation describe-stacks --stack-name petStoreStack --region us-east-1 --query 'Stacks[0].Outputs[0].{PetStoreAPI:OutputValue}'*

## Exercise 13.2: Edit the HTML Files

In steps 1 through 5, you are going to update the URL inside your .html files to point to the Amazon API Gateway stage that you have created. You do this so that your web application (.html files) knows the endpoint where to send your pet data.

1. Open index.html in the project folder and locate line 68 to find the variable named api_gw_endpoint. Input the value you retrieved from the previous command in Exercise 13.1.

var api_gw_endpoint = "https://cdvhqasdfnk444fe.execute-api.us-east-1
.amazonaws.com/PetStoreProd/"

2. Open pets.html.
3. Input the value you received from the last command on line 96, and add /pets to the end of the string:

var api_gw_endpoint = "https://cdvhqasdfnk444fe.execute-api.us-east-
1.amazonaws.com/PetStoreProd/pets"

4. Open add-pet.html.
5. Input the value you received from the last command on line 87, and add /pets to the end.

var api_gw_endpoint = "https://cdvhqasdfnk444fe.execute-api.us-east-1
.amazonaws.com/PetStoreProd/pets"

6. Create a new Amazon S3 bucket for your website.

aws s3 mb s3://my-bucket-name --region us-east-1

7. Copy the project files to the website.
8. aws s3 cp . s3://my-bucket-name --recursive

aws s3 rm s3://my-bucket-name/sam –recursive

Here you are uploading all the files from your project folder to Amazon S3 and then removing the SAM template from the bucket. You do not want others to have access to your template files and AWS Lambda functions. You want others to have access only to the end application.

9. Change the Amazon S3 bucket name inside of the policy.json to your bucket name. This will be on line 12.
10. Enable public read access for the bucket:
11. aws s3api put-bucket-policy --bucket my-bucket-name --policy file://policy.json

If successful, this command will not return any information. You are enabling the Amazon S3 bucket to be publicly accessible, meaning that everyone can access your website.

12. Enable the static website.

aws s3 website s3://my-bucket-name/ --index-document index.html --error-document index.html

The Amazon S3 bucket now acts as a web server and is running your pet store application.

13. Navigate to the website.

url: http://my-bucket-name.s3-website-us-east-1.amazonaws.com/index.html

14. Navigate Amazon API Gateway, AWS Lambda, Amazon DynamoDB, and the AWS SAM template to view the configuration.

Now that the application has been deployed, you can view all the individual components inside the AWS Management Console.

Inside Amazon API Gateway, you should see the PetStoreAPIGW. If you review the resources, you will see the various HTTP methods that you are allowing for your API.

In AWS Lambda, two functions were created: savePet for saving pets to Amazon DynamoDB and getPets for retrieving pets stored in Amazon DynamoDB.

In Amazon DynamoDB, you should have a table called PetStore. You can view the items in this table, though by default there should be none. After you create your first pet, however, you will be able to see some items in the table.

You can view the AWS SAM template and the AWS CloudFormation stack to see exactly how each of these resources were created.

## Exercise 13.3: Define an AWS SAM Template

In this exercise, you will develop an AWS Lambda function locally and then test that Lambda function using the AWS SAM CLI. To perform this exercise successfully, you must have AWS SAM CLI installed. For information on how to install the AWS SAM CLI, review the following documentation: https://github.com/awslabs/aws-sam-cli. The following steps assume that you have a working AWS SAM CLI installation.

1. Once you have installed AWS SAM CLI, open your favorite integrated development environment (IDE) and define an AWS SAM template.
2. Enter the following in your template file:
3. AWSTemplateFormatVersion: '2010-09-09'
4. Transform: AWS::Serverless-2016-10-31
5. 
6. Description: Welcome to the Pet Store Demo
7. 
8. Resources:
9.   PetStore:
10.     Type: AWS::Serverless::Function
11.     Properties:

12. Runtime: nodejs8.10

Handler: index.handler

13. Save the file as template.yaml.

You have created the SAM template and saved the file locally. In subsequent exercises, you will use this information to execute an AWS Lambda function.

## Exercise 13.4: Define an AWS Lambda Function Locally

Now that you have a valid SAM template, you can define your AWS Lambda function locally. In this example, use Nodejs 8.10, but you can use any AWS Lambda supported language.

1. Open your favorite IDE, and type the following Nodejs code:
2. 'use strict';
3. 
4. //A simple Lambda function
5. exports.handler = (event, context, callback) => {
6. 
7. console.log('This is our local lambda function');
8. console.log('Creating a PetStore service');
9. callback(null, "Hello " + event.Records[0].dynamodb.NewImage.Message.S + "! What kind of pet are you interested in?");

}

10. Save the file as index.js.

You have two files: an index.js and the SAM template. In the next exercise, you will generate an event source that will be used as the trigger for the AWS Lambda function.

## Exercise 13.5: Generate an Event Source

Now that you have a valid SAM template and a valid AWS Lambda Nodejs 8.10 function, you can generate an event source.

1. Inside your terminal, type the following to generate an event source:

sam local generate-event dynamodb update > event.json

This will generate an Amazon DynamoDB update event. For a list of all of the event sources, type the following:

sam local generate-event –help

2. Modify the event source JSON file (event.json). On line 17, change New Item! to your first and last names.

"S": "John Smith"

You have now configured the three pieces that you need: the AWS SAM template, the AWS Lambda function, and the event source. In the next exercise, you will be able to run the AWS Lambda function locally.

## Exercise 13.6: Run the AWS Lambda Function

Trigger and execute the AWS Lambda function.

1. In your terminal, type the following to execute the AWS Lambda function:

sam local invoke "PetStore" -e event.json

You will see the following message:

*Hello Casey Gerena! What kind of pet are you interested in?*

The AWS Lambda Docker image is downloaded to your local environment, and the event.json serves as all of the data that will be received as an event source to the AWS Lambda function. Inside the AWS SAM template, you will have given this function the name PetStore; however, you can define as many functions as you need to in order to build your application.

## Exercise 13.7: Modify the AWS SAM template to Include an API Locally

To make your pet store into an API, modify the template.yaml.

1. Open the template.yaml file, and modify it to look like the following:
2. AWSTemplateFormatVersion: '2010-09-09'
3. Transform: AWS::Serverless-2016-10-31
4. 
5. Description: Welcome to the Pet Store Demo
6. 
7. Resources:
8.   PetStore:
9.     Type: AWS::Serverless::Function
10.     Properties:
11.       Runtime: nodejs8.10

12. Handler: index.handler
13. Events:
14. PetStore:
15. Type: Api
16. Properties:
17. Path: /

    Method: any
18. Save the template.yaml file.

You have modified the AWS SAM template to connect an Amazon API Gateway event for any method (GET, POST, and so on) to the AWS Lambda function. In the next exercise, you will modify the AWS Lambda function to work with the API.

## Exercise 13.8: Modify Your AWS Lambda Function for the API

After you have defined an API, modify your AWS Lambda function.

1. Open the index.js file, and make the following changes:
2. 'use strict';
3. 
4. //A simple Lambda function
5. exports.handler = (event, context, callback) => {
6. 
7. console.log('DEBUG: This is our local lambda function');
8. console.log('DEBUG: Creating a PetStore service');
9. 
10. callback(null, {
11. statusCode: 200,
12. headers: { "x-petstore-custom-header": "custom header from petstore service" },
13. body: '{"message": "Hello! Welcome to the PetStore. What kind of Pet are you interested in?"}'
14. })
15.

}

16. Save the index.js file.

- You have modified the AWS Lambda function to respond to an API REST request. However, you have not actually executed anything—you will do that in the next exercise.

## Exercise 13.9: Run Amazon API Gateway Locally

Now that you have everything defined, run Amazon API Gateway locally.

1. Open a terminal and type the following:

```
sam local start-api
```

You will see output that looks like the following. Take note of the URL.

```
2018-10-11 23:05:25 Mounting PetStore at http://127.0.0.1:3000/hello [GET]

2018-10-11 23:05:25 You can now browse to the above endpoints to invoke your functions. You do not need to restart/reload SAM CLI while working on your functions changes will be reflected instantly/automatically. You only need to restart SAM CLI if you update your AWS SAM template

2018-10-11 23:05:25  * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
```

2. Open a web browser, and navigate to the previous URL.

- You will see the following message:

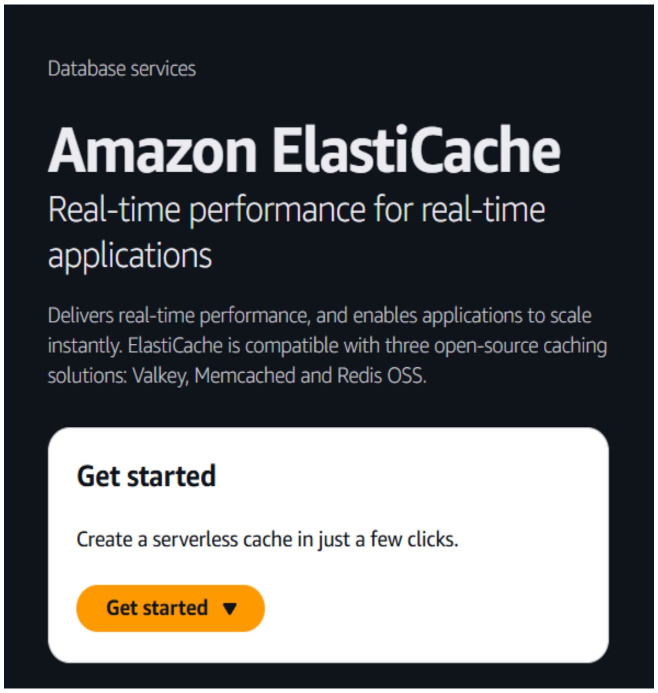
```
Message: "Hello! Welcome to the Pet Store. What kind of Pet are you interested in?"
```

- When you navigate to the URL, the local API Gateway forwards the request to AWS Lambda, which is also running locally, provided by index.js. You can now build serverless applications locally. When you are ready to deploy to a development or production environment, deploy the serverless applications to the AWS Cloud with AWS SAM. This allows developers to iterate through their code quickly and make improvements locally.

# Chapter 14: ElastiCache and MemCached

## Exercise 14.1: Create an Amazon ElastiCache Cluster that runs Memcached

1. Sign in to the AWS Management Console, and open the ElastiCache console at https://console.aws.amazon.com/elasticache/.



Database services

Amazon ElastiCache

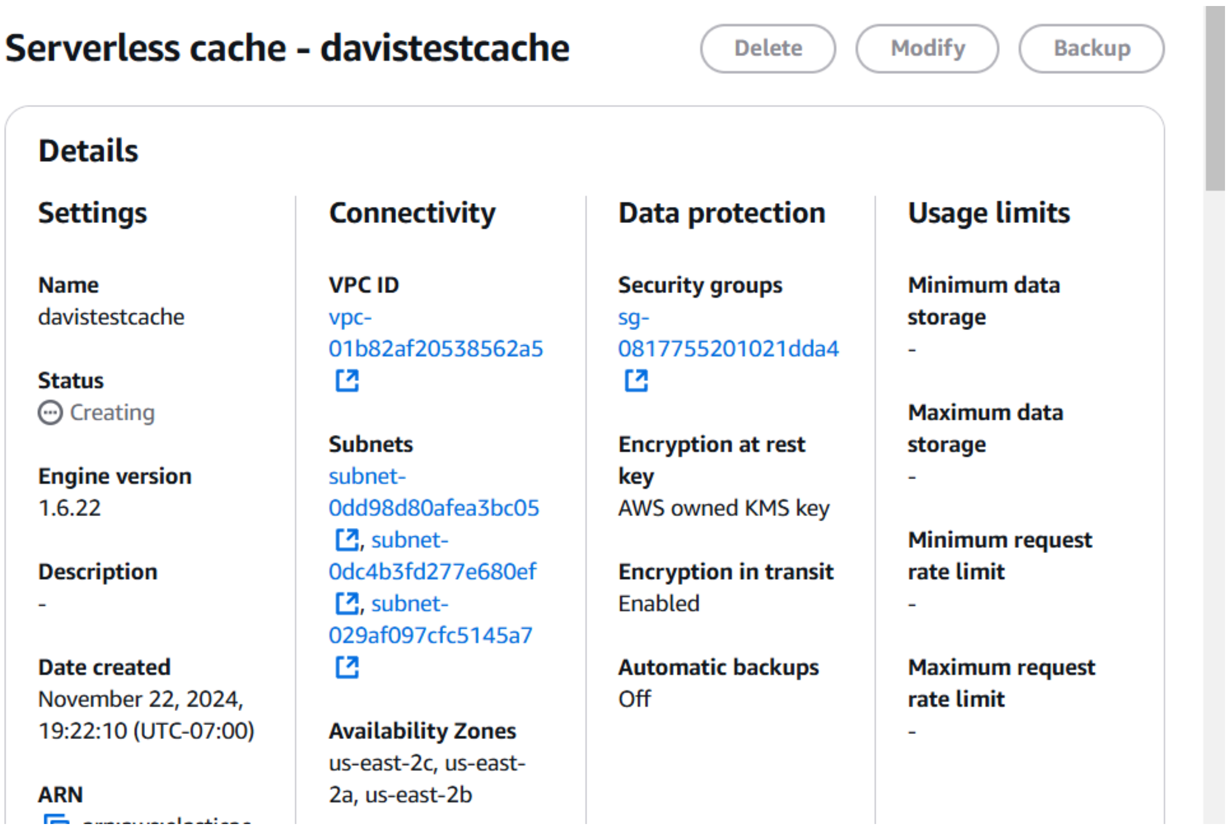
Real-time performance for real-time applications

Delivers real-time performance, and enables applications to scale instantly. ElastiCache is compatible with three open-source caching solutions: Valkey, Memcached and Redis OSS.

Get started

Create a serverless cache in just a few clicks.

2. To create a new ElastiCache cluster, begin the launch and configuration process. (You must have 3 vpc's to create a cache)
3. For **Cluster engine**, choose **Memcached** and configure the cluster name, number of nodes, and node type.
4. (Optional) Configure the security group and maintenance window as needed.
5. Review the cluster configuration and begin provisioning the cluster. Connect to the cluster with any Memcached client by using the DNS name of the cluster.
6. You should be greeted with a screen similar to below


Serverless cache - davistestcache

Details

| Settings | Connectivity | Data protection | Usage limits |
|---|---|---|---|
| **Name** davistestcache | **VPC ID** vpc-01b82af20538562a5 | **Security groups** sg-0817755201021dda4 | **Minimum data storage** - |
| **Status** Creating | **Subnets** subnet-0dd98d80afea3bc05, subnet-0dc4b3fd277e680ef, subnet-029af097cfc5145a7 | **Encryption at rest key** AWS owned KMS key | **Maximum data storage** - |
| **Engine version** 1.6.22 | **Availability Zones** us-east-2c, us-east-2a, us-east-2b | **Encryption in transit** Enabled | **Minimum request rate limit** - |
| **Description** - | | **Automatic backups** Off | **Maximum request rate limit** - |
| **Date created** November 22, 2024, 19:22:10 (UTC-07:00) | | | |
| **ARN** | | | |

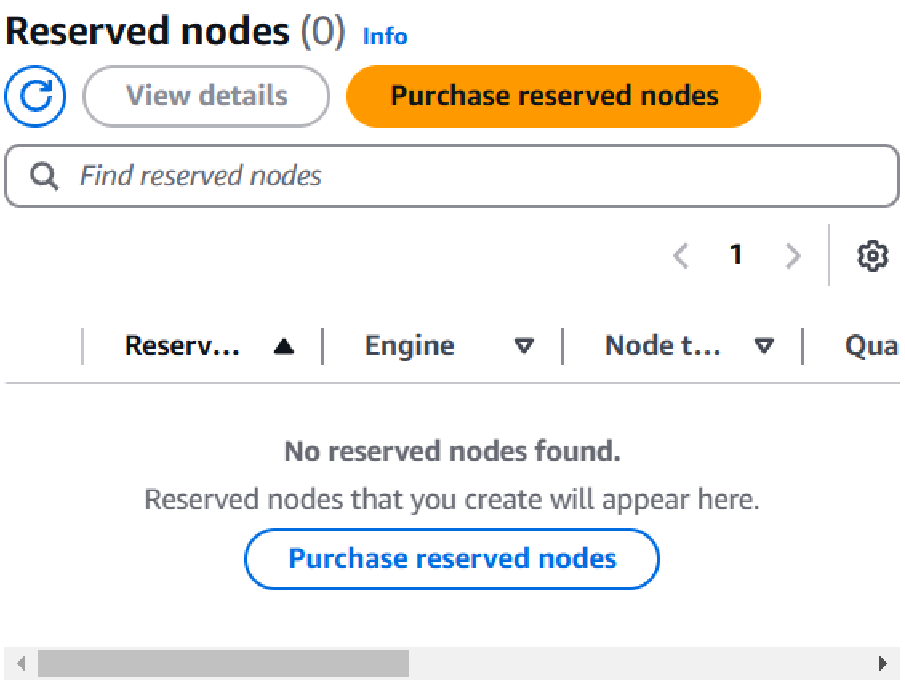
You have now created the Elasticache.

## Exercise 14.2: Expand the size of a Memcached Cluster

In this exercise, you will expand the size of an existing cluster inside your elasticache.

1. Launch a Memcached cluster by following the steps in the previous exercise.
2. Navigate to the Amazon ElastiCache dashboard, and view the configuration of your existing cluster.

3. Under reserved nodes, you will need to purchase nodes for your cache



No reserved nodes found.

Reserved nodes that you create will appear here.

4. Apply the changes to the configuration, and wait for the new node to finish provisioning.
5. Confirm that the new node has been provisioned, and connect to the node using a Memcached client.

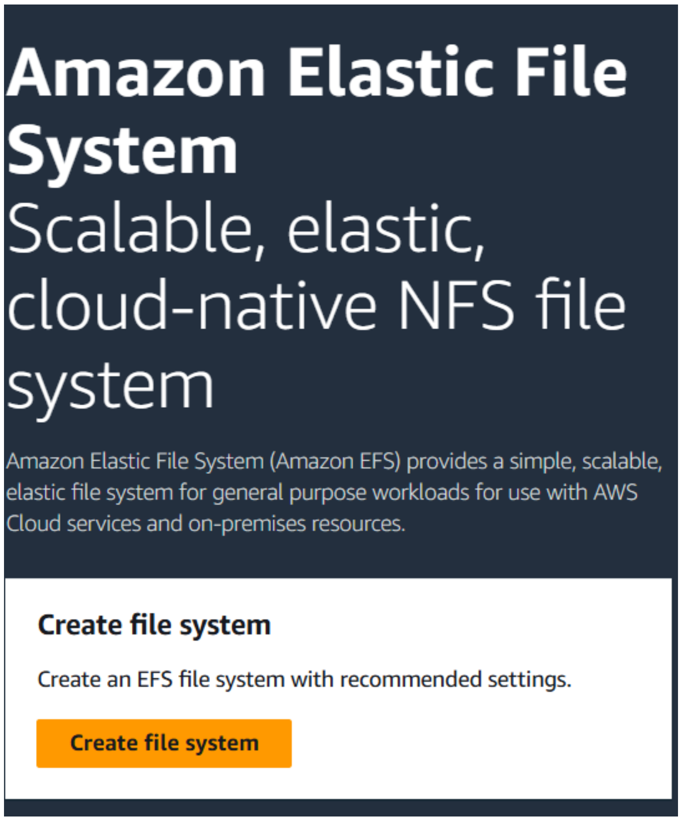
   You have horizontally scaled an existing ElastiCache cluster by adding a new cache node.

## Exercise 14.3: Create and Attach an Amazon EFS Volume

In this exercise, you will create a new Amazon EFS volume and attach it to a running instance.

1. While signed in to the AWS Management Console, open the Amazon EC2 console at https://console.aws.amazon.com/ec2.

   If you don't see a running Linux instance, launch a new instance.
2. Open the Amazon EFS service dashboard. Choose Create File System.



Amazon Elastic File System

Scalable, elastic, cloud-native NFS file system

Amazon Elastic File System (Amazon EFS) provides a simple, scalable, elastic file system for general purpose workloads for use with AWS Cloud services and on-premises resources.

Create file system

Create an EFS file system with recommended settings.

3. Select the Amazon VPC where your Linux instance is running.
4. Accept the default mount targets, and make a note of the security group ID assigned to the targets.
5. Choose any settings, and then create the file system.
6. Assign the same default security group used by the file system to your Linux instance.
7. Log in to the console of the Linux instance, and install the NFS client on the Amazon EC2 instance. For Amazon Linux, use the following command:

   sudo yum install –y nfs-utils

8. Create a new directory on your Amazon EC2 instances, such as awsdev: sudo mkdir awsdev.
9. Mount the file system using the DNS name:

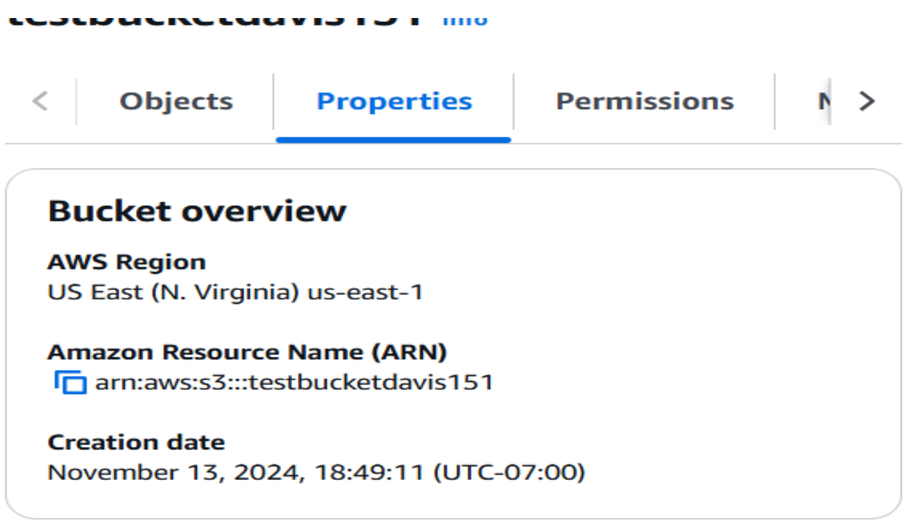
   sudo mount –t nfs4 –o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600, retrains=2 fs-12341234.efs.region-1.amazonaws.com:/ awsdev

   You have mounted the Amazon EFS volume to the instance.

## Exercise 14.4: Create and Upload to an Amazon S3 Bucket

In this exercise, you will create an Amazon S3 bucket and upload and publish files of the bucket.

1. While signed in to the AWS Management Console, open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.
3. On the Name and region page, enter a globally unique name for your bucket and select the appropriate AWS Region. Accept all of the remaining default settings.
4. Choose Create the bucket.



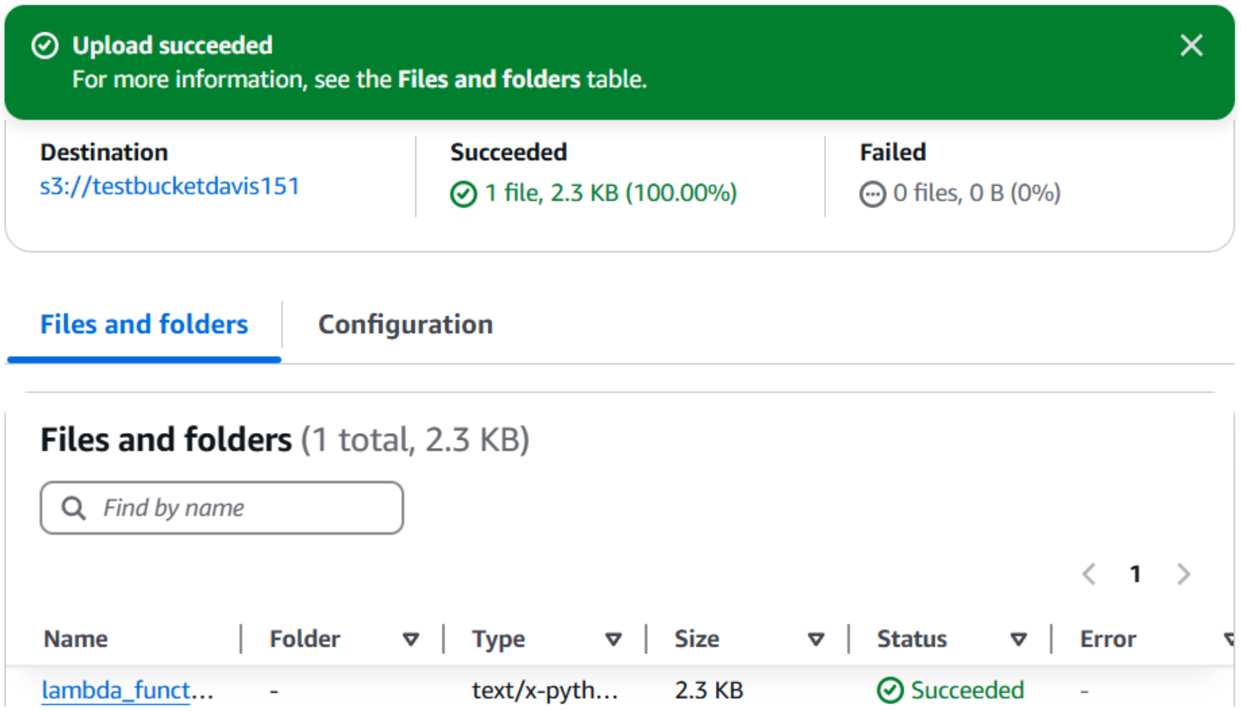
Bucket overview

AWS Region
US East (N. Virginia) us-east-1

Amazon Resource Name (ARN)
arn:aws:s3:::testbucketdavis151

Creation date
November 13, 2024, 18:49:11 (UTC-07:00)

5. Select your bucket.
6. Upload data files to the new Amazon S3 bucket:
   1. Choose Upload.
   2. In the Upload - Select Files wizard, choose Add Files and choose a file that you want to share publicly.
   3. Choose Next.
7. To make the file available to the general public, on the Set Permissions page, under Manage Public Permissions, grant everyone read access to the object.
8. Review and upload the file.
9. Select the object name to go to the properties screen. Select and open the URL for the file in a new browser window


Upload succeeded
For more information, see the **Files and folders** table.

| Destination | Succeeded | Failed |
|---|---|---|
| s3://testbucketdavis151 | 1 file, 2.3 KB (100.00%) | 0 files, 0 B (0%) |

Files and folders (1 total, 2.3 KB)

| Name | Folder | Type | Size | Status | Error |
|---|---|---|---|---|---|
| lambda_funct… | - | text/x-pyth… | 2.3 KB | Succeeded | - |

.

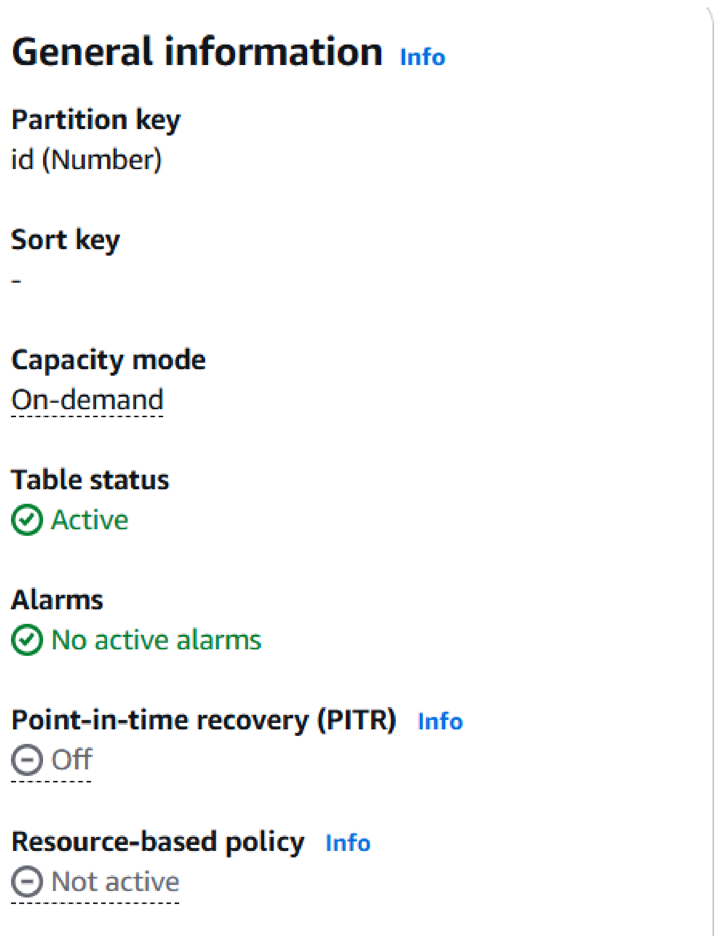
You should see your file in the S3 bucket.

## Exercise 14.5: Create an Amazon DynamoDB Table

In this exercise, you will create an DynamoDB table.

1. While signed in to the AWS Management Console, open the DynamoDB console at https://console.aws.amazon.com/dynamodb/.
2. Choose **Create Table and then do the following:**
    1. In **Table**, enter the table name.
    2. For **Primary key**, in the **Partition** field, type **Id**.
    3. Set the **data type** to String.
3. Retain all the remaining default settings and choose **Create**.

You have created a DynamoDB table. You should be greeted with the below


General information Info

Partition key
id (Number)

Sort key
-

Capacity mode
On-demand

Table status
✓ Active

Alarms
✓ No active alarms

Point-in-time recovery (PITR) Info
Off

Resource-based policy Info
Not active

## Exercise 14.6: Enable Amazon S3 Versioning

In this exercise, you will enable Amazon S3 versioning, which prevents objects from being accidentally deleted or overwritten.

1. While signed in to the AWS Management Console, open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the **Bucket name** list, choose the name of the bucket for which you want to enable versioning.

- If you don't have a bucket, follow the steps in Exercise 14.4 to create a new bucket.

1. Choose **Properties**.
2. Choose **Versioning**.
3. Choose **Enable versioning**, and then choose **Save**.

Your bucket is now versioning enabled.

## Exercise 14.7: Create an Amazon DynamoDB Global Table

In this exercise, you will create a DynamoDB global table.

1. While signed in to the AWS Management Console, open the Amazon S3 console at https://console.aws.amazon.com/dynamodb.
2. Choose a region for the source table for your DynamoDB global table.
3. In the navigation pane on the left side of the console, choose **Create Table** and then do the following:
    1. For **Table name**, type **Tables**.
    2. For **Primary key**, choose an appropriate primary key. Choose **Add sort key**, and type an appropriate sort key. The data type of both the partition key and the sort key should be strings.
4. To create the table, choose **Create**.

- This table will serve as the first replica table in a new global table, and it will be the prototype for other replica tables that you add later.

5. Select the **Global Tables** tab, and then choose **Enable streams**. Leave the **View type** at its default value (New and old images).

6. Choose **Add region**, and then choose another region where you want to deploy another replica table. In this case, choose **US West (Oregon)** and then choose **Continue**. This will start the table creation process in US West (Oregon).

- The console will check to ensure that there is no table with the same name in the selected region. (If a table with the same name does exist, then you must delete the existing table before you can create a new replica table in that region.)
- The **Global Table** tab for the selected table (and for any other replica tables) will show that the table is replicated in multiple regions.

7. Add another region so that your global table is replicated and synchronized across the United States and Europe. To do this, repeat step 6, but this time specify **EU (Frankfurt)** instead of **US West (Oregon)**.

You have created a DynamoDB global table.

## Exercise 14.8: Enable Cross-Region Replication

In this exercise, you will enable cross-region replication of the contents of the original bucket to a new bucket in a different region.

1. While signed into the AWS Management Console, open the Amazon S3 console at https://console.aws.amazon.com/s3/.

2. Create a new bucket in a different region from the bucket that you created in Exercise 14.4. Enable versioning on the new bucket (see Exercise 14.6).
3. Choose **Management**, choose Replication, and then choose **Add rule**.
4. In the **Replication rule** wizard, under **Set Source**, choose **Entire Bucket**.
5. Choose **Next**.
6. On the **Set destination** page, under **Destination bucket**, choose your newly-created bucket.
7. Choose the storage class for the target bucket. Under **Options**, select **Change the storage class for the replicated objects**. Select a storage class.
8. Choose **Next**.
9. For IAM, on the **Configure options** page, under **Select** role, choose **Create new role**.
10. Choose **Next**.
11. Choose **Save**.
12. Load a new object in the source bucket.

- The object appears in the target bucket.

  You have enabled cross-region replication, which can be used for compliance and disaster recovery.

## Exercise 14.9: Create an Amazon DynamoDB Backup Table

In this exercise, you will create a DynamoDB table backup.

1. While signed into the AWS Management Console, open the DynamoDB console at https://console.aws.amazon.com/dynamodb/.
2. Choose one of your existing tables. If there are no tables, follow the steps in Exercise 14.7 to create a new table.
3. On the **Backups** tab, choose **Create Backup**.
4. Type a name for the backup name of the table you are backing. Then choose **Create** to create the backup.

- While the backup is being created, the backup status is set to Creating. After the backup is finalized, the backup status changes to Available.

  You have created a backup of a DynamoDB table.

## Exercise 14.10: Restoring an Amazon DynamoDB Table from a Backup

In this exercise, you will restore a DynamoDB table by using the backup created in the previous exercise.

1. While signed in to the AWS Management Console, navigate to the DynamoDB console at https://console.aws.amazon.com/dynamodb/.
2. In the navigation pane on the left side of the console, choose **Backups**.
3. In the list of backups, choose the backup that you created in the previous step.
4. Choose **Restore Backup**.
5. Type a table name as the new table name. Confirm the backup name and other backup details. Then choose **Restore table** to start the restore process.

- The table that is being restored is shown with the status Creating. After the restore process is finished, the status of your new table changes to Active.

You have performed the restoration of a DynamoDB table from a backup.

# Chapter 15: Monitoring and Troubleshooting

## Exercise 15.1: Create a CloudWatch Alarm for an S3 Bucket.

It is common to monitor the storage usage of your Amazon S3 buckets and trigger notifications when there is a large increase in storage used. In this exercise, you will use the AWS CLI to configure an Amazon CloudWatch alarm to trigger a notification when more than 1 KB of data is uploaded to an Amazon S3 bucket.

If you need directions while completing this exercise, see "Using Amazon CloudWatch Alarms" here:

*https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html*

1. Create an Amazon S3 bucket in your AWS account. For instructions, see this page:

https://docs.aws.amazon.com/AmazonS3/latest/user-guide/ create-bucket.html

2. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. Select Alarms ➢ Create Alarm.
4. Choose Select Metric.
    1. Select the All Metrics tab.
    2. Expand AWS Namespaces.
    3. Select S3.
    4. Select Storage Metrics.
    5. Select a metric where BucketName matches the name of the Amazon S3 bucket that you created and where Metric Name is BucketSizeBytes.
5. Choose Select Metric.
6. Under Alarm Details:
    1. For Name, enter **S3 Storage Alarm**.
    2. For the comparator, select >= (greater than or equal to).
    3. Set the value to 1000 for 1 KB.
7. Under Actions:
    1. For Whenever This Alarm, select State Is ALARM.
    2. For Send Notification To, select New List.

   3. For Name, enter **My S3 Alarm List**.
   4. For Email List, enter your email address.

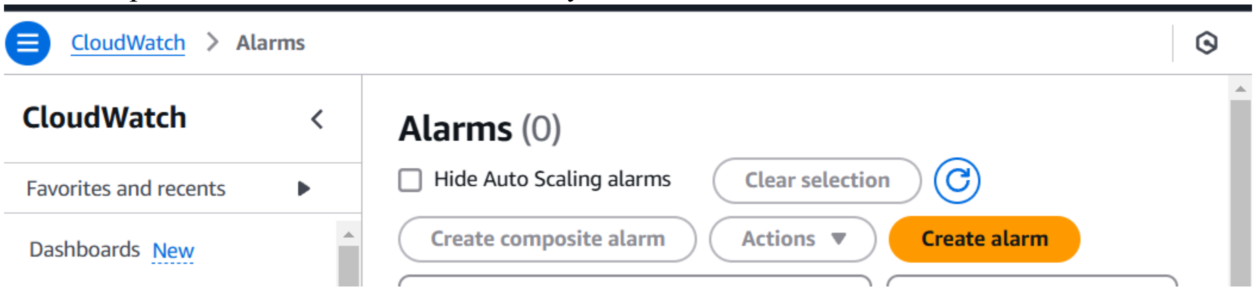
8. Choose Create Alarm.

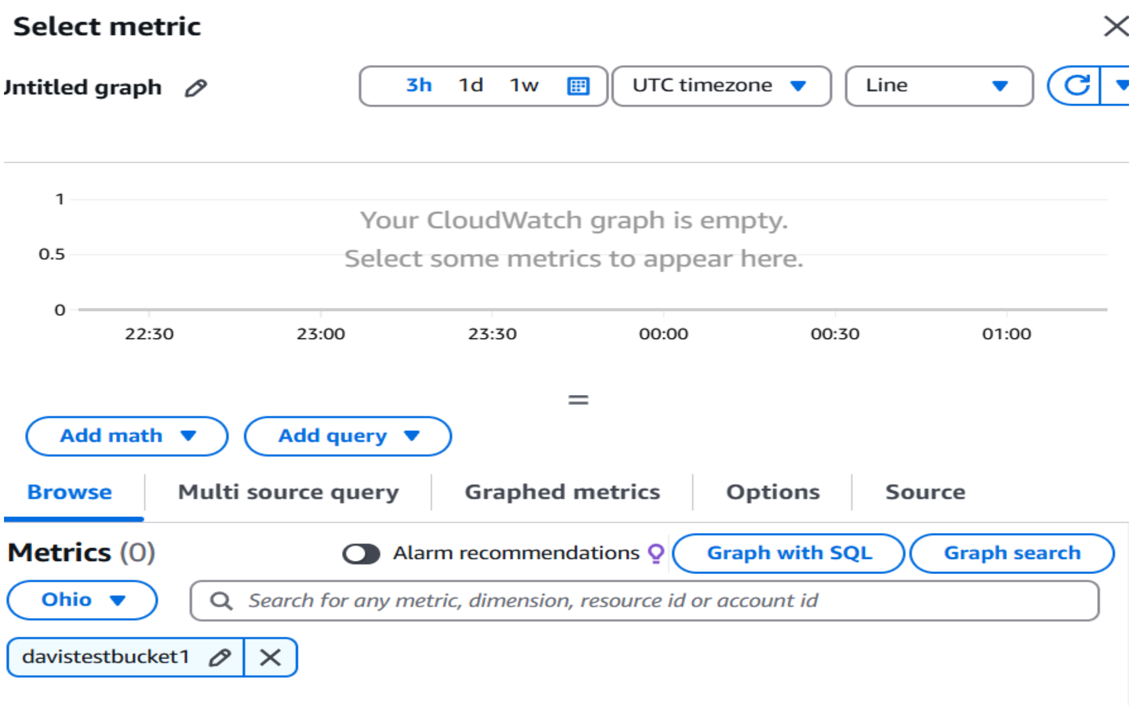
The alarm is created in your account. If you already have data in your Amazon S3 bucket, it is switched from Insufficient Data to Alarm state. Otherwise, try uploading several files to your bucket to monitor changes in alarm state.

To delete the alarm, follow these steps:

1. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Select Alarms.
3. Select the alarm you want to delete.
4. For Actions, select Delete.

In this exercise, you created an Amazon CloudWatch alarm to notify administrators when large files are uploaded to Amazon S3 buckets in your account.





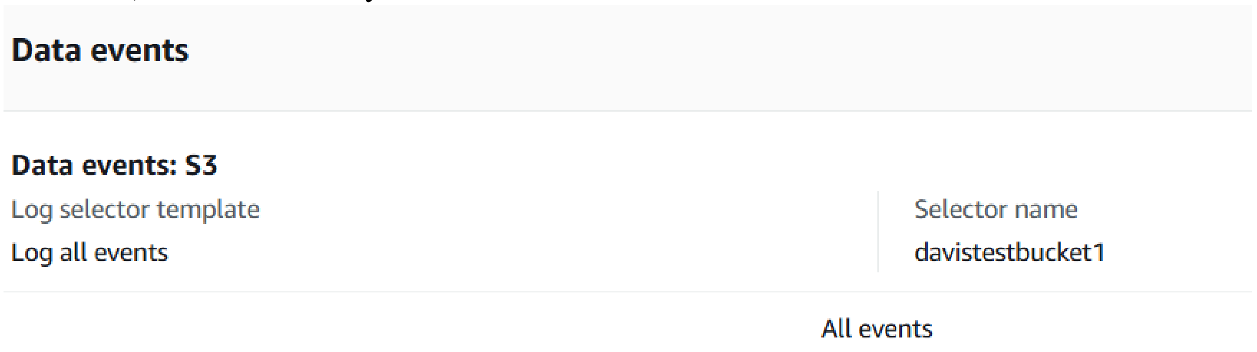
## Exercise 15.2: Enable AWS CloudTrail on an S3 Bucket.

1. In this exercise, you will set up access logs to an Amazon S3 bucket in your account to monitor activity.Create an Amazon S3 bucket in your AWS account.

For instructions on how to do so, see the following:

https://docs.aws.amazon.com/AmazonS3/latest/user-guide/ create-bucket.html

2. Open the AWS CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
3. Select Create Trail.
4. Set Trail name to s3_logs.
5. Under Management Events, select None.
6. Under Data Events, select Add S3 Bucket.
7. For S3 bucket, enter your Amazon S3 bucket name.
8. Under Storage Location, for Create A New S3 bucket, select Yes.
9. For Name, enter a name for your Amazon S3 bucket.



Data events

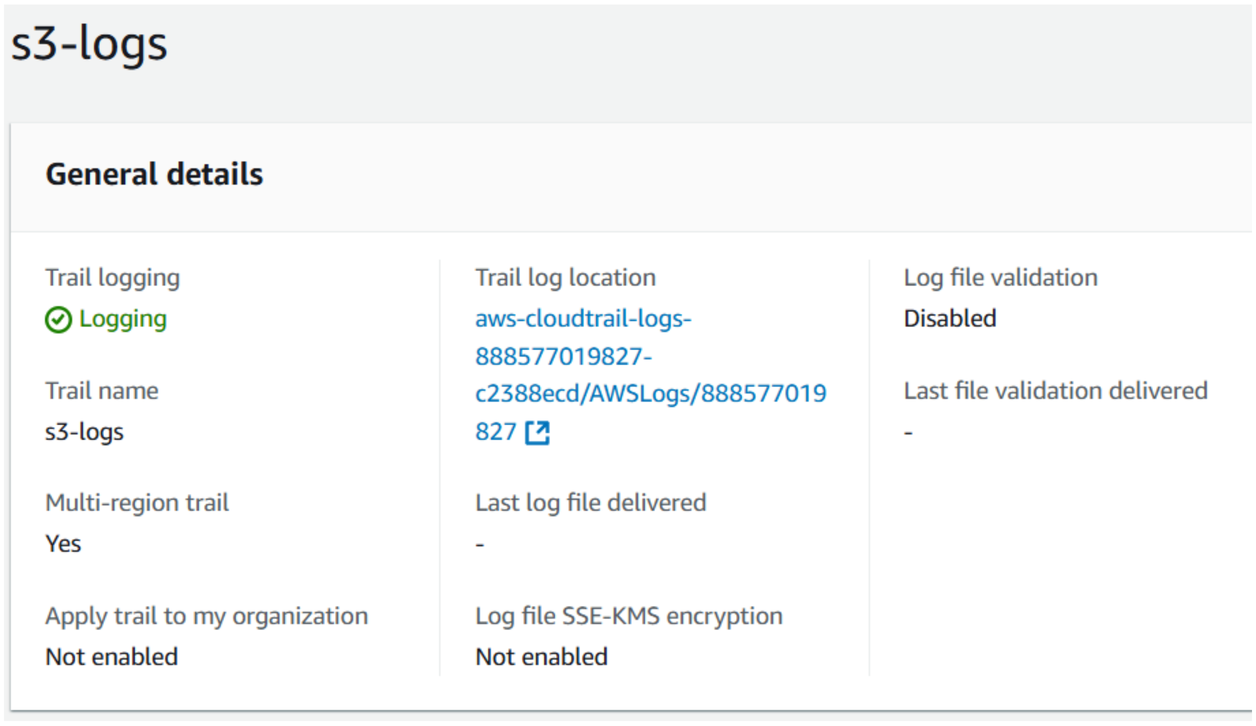
Data events: S3

| Log selector template | Selector name |
|---|---|
| Log all events | davistestbucket1 |

All events

10. Choose Create.

In this exercise, you enabled AWS CloudTrail to record data events and store corresponding logs to an Amazon S3 bucket.



s3-logs

General details

| | | |
|---|---|---|
| Trail logging<br>✓ Logging | Trail log location<br>aws-cloudtrail-logs-888577019827-c2388ecd/AWSLogs/888577019827 | Log file validation<br>Disabled |
| Trail name<br>s3-logs | Last log file delivered<br>- | Last file validation delivered<br>- |
| Multi-region trail<br>Yes | Log file SSE-KMS encryption<br>Not enabled | |
| Apply trail to my organization<br>Not enabled | | |

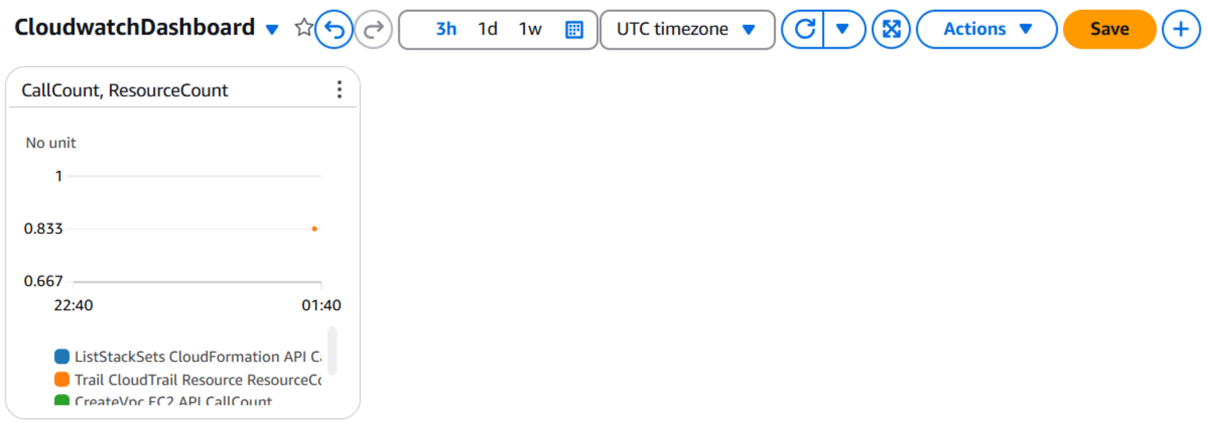
## Exercise 15.3: Amazon CloudWatch Dashboard

In this exercise, you will create an Amazon CloudWatch dashboard to see graphed metric data.

1. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, select Dashboards.
3. Choose Create Dashboard.
4. For Dashboard Name, enter a name for your dashboard.
5. Select Create Dashboard.
6. In the modal window, select the Line graph.
7. Choose Configure.
8. From the available metrics, select one or more metrics that you want to monitor.
9. Choose Create Widget.
10. To add more widgets, choose Add Widget and repeat steps 6 through 9 for other widget types.
11. Choose Save Dashboard.

In this exercise, you created an Amazon CloudWatch dashboard to create graphs of important metric data for resources in your account.

Basic Dashboard Monitoring CloudTrail itself, EC2 Call count, and VPC creation:



## References

Alteen, N., Fisher, J., Gerena, C., Gruver, W., Jalis, A., Osman, H., Pagan, M., Patlolla, S., & Roth, M. (2019). *AWS Certified Developer Official Study Guide, associate exam*. Sybex.

Welcome to AWS documentation. (n.d.). https://docs.aws.amazon.com/