CMPT 489

Assignment 8

Marcelo Ollin Paco Zepeda
301180252

Table of Contents

Part 1: Amazon IAM

Task 1:

According to your opinion is it a good practice to create multiple users in AWS? Justify your answer and give examples.

Yes, it is a good practice to create multiple users in AWS. Doing so achieves the security objectives of integrity, confidentiality and availability (CIA Triad). As mentioned in the lectures, confidentiality is the avoidance of the unauthorized disclosure of information. Creating a user with a need to know basis allows for the protection of data, providing access for those who are allowed to view it while disallowing others from learning anything about its content. Moreover, the creation of multiple users with different privileges ensures that information is not altered in an unauthorized way. It also allows for information to be accessible and modifiable in a timely fashion by those who are authorized to do so.

Task 2:

What could be a use case for an IAM *role*?

According to the Amazon web services, IAM roles are a secure way to grant permissions to entities that you trust. The IAM rules issue keys that are valid for short durations, making them a more secure way to grant access. With that being said, a potential use case for an IAM role would be for an IAM user in another account.

Task 3:

What is the difference between an IAM *role* and an IAM *policy*?

IAM roles define the set of permissions needed for making an AWS service request, whereas IAM policies defines the permission required to make such request. Moreover, an IAM role does not have any credentials and cannon make direct request to AWS services. IAM roles are meant to be assumed by authorized entities, such as IAM users, applications, or an AWS service such as EC2.

Part 2: Amazon EC2 & DynamoDB

Task 4:

What is needed in order for the EC2 instance to be able to access the newly created DynamoDB table?

There is a series of requirements needed to access the newly created DynamoDB table. The EC2 instance needs to have pip3 and awscli installed. Once pip3 is installed, the boto3 package need to be downloaded in order to use the DynamoDB api. In order for the python3 program to communicate with our DynamoDB instance in AWS, we need to set up our AWS credentials in the EC2 instance using the **aws configure** command that comes with the awscli utility. In task 5 there will be an in-detail explanation on how to carry out these steps.

Task 5:

Report the steps you took in order to EC2 instance access the DynamoDB table.

Here are the steps taken in order for the EC2 instance to access the DynamoDB table (Assuming already logged on the EC2 instance):

```
sudo apt install python3-pip
```

If the command above generates the following error:

```
Reading package lists... Done
Building dependency tree Reading state information... Done
```

```
E: Unable to locate package python3-pip
```

Then you need to modify the /etc/apt/sources.list file to add universe at the end of the following lines:

```
deb http://archive.ubuntu.com/ubuntu bionic main universe
deb http://archive.ubuntu.com/ubuntu bionic-security main universe
deb http://archive.ubuntu.com/ubuntu bionic-updates main universe
```

Once that's done, run the following commands:

```
sudo apt update
sudo apt install python3-pip
```

Now that pip3 is installed, we need to download the boto3 package:

```
pip3 install boto3
```

In order for us to communicate with DynamoDB, we need to configure our AWS credentials:

```
$ aws configure
AWS Access Key ID [None]: AKIAWKKBIXXZVNGUEVRA
AWS Secret Access Key [None]: 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
Default region name [None]:  us-east-1
Default output format [None]: json
```

The command above creates the following files:

```
~/.aws/credentials
~/.aws/config
```

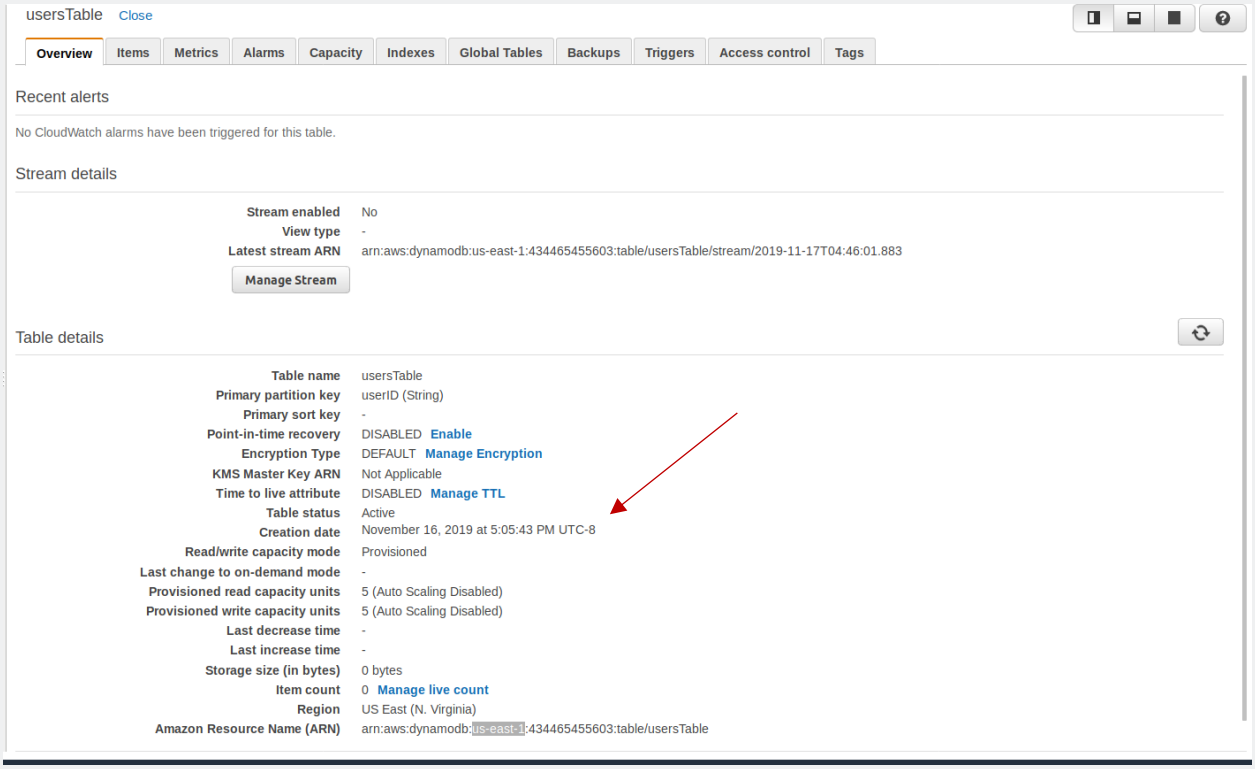
Now, we create a python3 program to communicate with the usersTable we created:

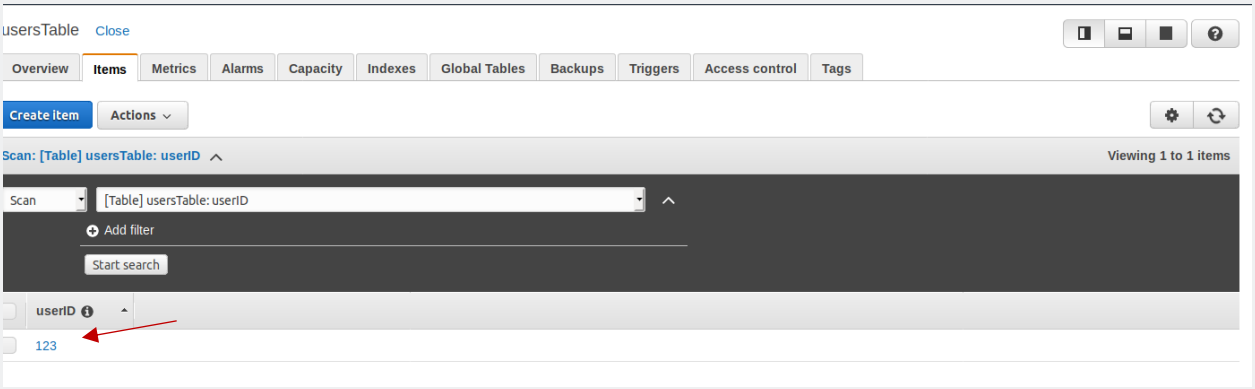
```
# Code source https://boto3.amazonaws.com/v1/documentation/api/latest/guide/dynamodb.html
import boto3

# Get the service resource.
dynamodb = boto3.resource('dynamodb')

# Instantiate a table resource object without actually
# creating a DynamoDB table. Note that the attributes of this table
# are lazy-loaded: a request is not made nor are the attribute
# values populated until the attributes
# on the table resource are accessed or its load() method is called.
table = dynamodb.Table('usersTable')

# Print out some data about the table.
# This will cause a request to be made to DynamoDB and its attribute
# values will be set based on the response.
print(table.creation_date_time)
print(table.item_count)
```

The program outputs the following:

```
2019-11-17 01:05:43.444000+00:00
1
```

The first line in the output above is the date and time when the table was created, in UNIX epoch time format and the second line is the number of items in the usersTable. Here are some screenshot verifying that this is in fact the DynamoDB instance recently created:





Part 3: Running a Web App on EC2

Task 6:

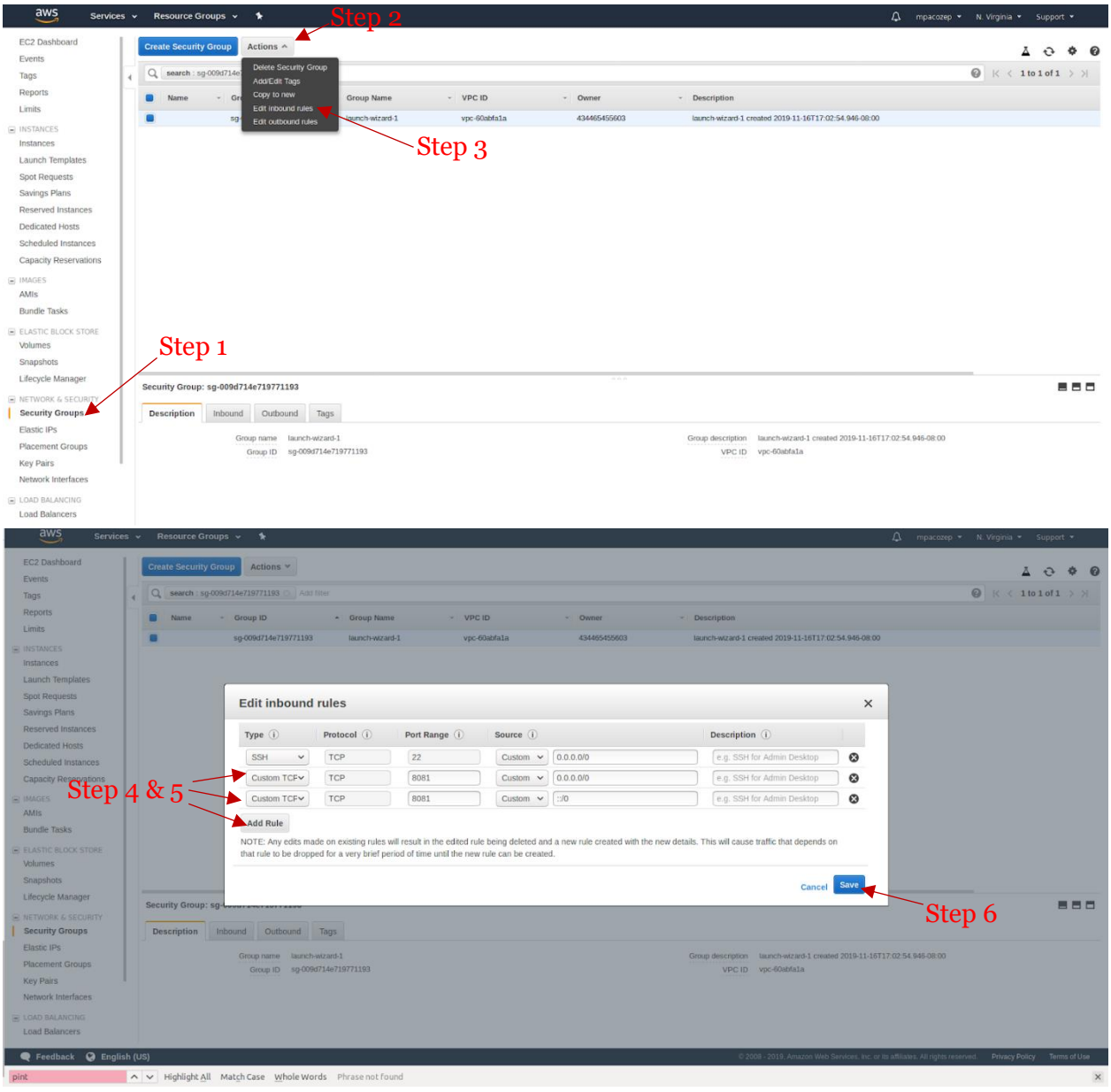
If you go to EC2 > Instances and click on the Instance you have created, then you will notice that there is a plethora of information about your newly created machine. There is an IPv4 Public IP created for your EC2 instance. If you right click and Stop the machine and then Start it again, you will realize that the IP assigned to that machine is changed. Why is that? What would you do in order to give your machine an IP Address that persists through reboots?

The reason why the IP assigned to the EC2 instance changed after a reboot is because of dynamic host configuration protocol (DHCP). By default, it is set to give the EC2 instance a dynamic IP address. In order for the IP address to persists between reboots, we need to set up a static IP address for the EC2 instance. This can be done using a feature in AWS called **Elastic IP**. An Elastic IP address is a static IPv4 address designed for dynamic cloud computing. Moreover, an Elastic IP address is associated with the AWS account we created.

Task 7:

If you try to setup a Web server listening to the port 8081 inside your instance you will soon realize that it is not accessible from the outside world. What is the AWS component responsible for allowing traffic to be sent to port 8081? What steps would you take in order to make it accessible from the outside world?

The AWS component responsible for allowing traffic to be sent to port 8081, is the security group that the EC2 instance is part of. For my EC2 instance, the security group name is launch-wizard-1. Here are the steps required in order to make port 8081 accessible from the outside world:



The webapp on port 8081 should now be accessible from the outside world: