Cloud Computing Lab
Lab 1 - AWS, EC2, EBS and S3

Task a : AWS Management Console

Before getting into the details of this lab (and further labs) it is important to be familiar with the central hub/point of AWS , the AWS Management Console. Take a read in the link to understand what you can do with the console.

What is AWS ?

Working with the AWS Management Console

All AWS services can be accessed using the AWS management console, take a look around the console and the services and familiarise yourself with the environment!

Task b : Understanding and creating AWS EC2(Elastic Compute) virtual machines

One of the most important is the AWS EC2 (Elastic Compute) service. This provides the actual ***compute*** for your cloud applications and as any cloud service should be, scalable(hence the name elastic!). EC2 is a service that provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.In simpler words EC2 is none other than a virtual machine.

Understanding AWS EC2 - 4 minute video explaining AWS EC2

Every EC2 instance has the following properties/options/attributes that need to compulsorily be configured:

- Amazon Machine Image(AMI)
- Instance Type (The Instance Type usually depends on the use of the VM)
- Specific Instance Details such as network,subnets,start up scripts etc.
- Security Groups - These are essentially the firewalls to your instance, they control the access to your instance.

The following steps will help you in setting up a EC2 instance. Make sure you really understand how to create an EC2 instance as EC2 will be used throughout this course

1. Make sure you have your AWS Educate credentials ready, you would have received a mail from AWS Educate to register and sign up and then change your password.
2. Once mail is received, login to AWS Educate. Go to AWS Management Console page. Under 'Services' tab, click on EC2.
3. You'll reach the EC2 console page. Click Launch Instance.
4. Here you'll be presented with a list of Amazon Machine Images (AMI). They're essentially the virtual OSes that will run your server. We recommend you pick Ubuntu as you're likely to be familiar with its command line tools and package manager.
5. Now you'll be presented with a list of Instance Types. Each one has different machine specs like number of vCPUs, size of memory, storage volume type, etc. Pick one that is appropriate to your needs. For this a t2.micro(free tier eligible) is good enough.You can leave the default settings in the Configure Instance, Add Storage, and Add Tags pages, or configure it according to your needs.
6. On the Configure Security Group page, make sure you allow traffic on SSH and HTTP. Make sure to also install an SSH client on your local machine. This is a critical step whenever setting up an EC2 instance as this defines the access to your instance from the internet!
7. Create a cryptographic key pair and download the private key to your system. Then launch the instance. You may need to change the permissions of the PEM key.
    a. PEM Key Permissions - download the keyname.pem file, change permissions by executing **chmod 400 keyname.pem**
8. Change the instance name to your SRN
9. Learn about how to remotely login to another machine using SSH.
    a. SSH Basics
10. Find the public DNS for your instance. Use that and the private key to SSH into your instance. You'll need to figure out what the default username of your VM is, and how to point your SSH client to the private key.
    Execute **ssh -i keyname.pem ubuntu@<public ip>** to ssh into Public network
    a. SSH into AWS instances
    b. General prerequisites for connecting to your instance
11. To delete an instance select the instance, choose *Instance State* and in the dropdown select *Terminate Instance*.

Note: AWS Educate probably does not allow you to create all kinds of instances. Make sure it's EC2. And ensure your Region is "US-East"(Northern Virginia). AWS Educate does not work for any other region.

Your task is to create an EC2 instance with the following configurations/settings:

AMI	Ubuntu Server 20.04 LTS (HVM), SSD Volume Type
Instance Type	t2.micro
Instance Details	Default values (But try to understand what the different options are)
Storage	Default values

You may get a warning saying “Improve your instances' security. Your security group, launch-wizard-1, is open to the world”, you can ignore this for now.

1. After creating and getting into your EC2 instance, update the “apt” package, and install GCC compiler.
   Execute:
   sudo apt-get update
   sudo apt install build-essential
   sudo apt-get install manpages-dev

2. Write a simple C program and compile it.
   Execute:
   sudo nano hello.c
   gcc hello.c
   ./a.out

Task c : Understanding and using AWS EBS(Elastic Block Store)

Before getting into AWS EBS, it is important to understand what a block store is and how they are different from other types of cloud storage: What is Block Storage?

Think of EBS as the hard drive (SSD or HDD) that you have in your laptop.Amazon EBS offers persistent storage for Amazon Elastic Compute Cloud (Amazon EC2) instances. Amazon EBS volumes are network-attached and persist independently from the life of an instance. Amazon EBS volumes are highly available, highly reliable volumes that can be leveraged as an Amazon EC2 instance boot partition or attached to a running Amazon EC2 instance as a standard block device. Think of EBS as the hard drive (SSD or HDD) that you have in your laptop.

The following links will help you understand EBS in a good amount of detail:

Amazon Elastic Block Store (EBS) Overview

Amazon Elastic Block Store

- EBS Volume console can be accessed through the EC2 console. Go to the EC2 console , on the right hand side *Elastic Block Store -> Volumes.*
- To check the volumes attached to your instance, select the instance on the EC2 console, on the button you will see

Another important feature/concept of EBS is to create snapshots.You can back up the data on your Amazon EBS volumes to Amazon S3 by taking point-in-time snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved.But note that although the EBS snapshots are stored in S3 buckets, you don't have access to this bucket as AWS abstracts those details from you as all you need are the snapshots.

Amazon EBS snapshots - Amazon Elastic Compute Cloud

Your sub tasks are to:

1. Create an EC2 instance with the below mentioned specs. The instance name should again be your SRN.
2. Create an EBS volume with the below mentioned specs. Check in the EBS console, the volume you have just created should have an “available” status, indicating it is ready to be attached to an EC2 instance. Name the EBS volume as SRN_ebs
3. Now, *attach* the created EBS volume to the created EC2 instance. Remember every EC2 instance already has a *boot volume EBS* , what you are *attaching* is an additional volume.

   Attaching a volume to an EC2 instance is analogous to just connecting a bare hard drive to a system without any file system,information etc. It is only physically connected but the OS needs more than just a physical attachment , i.e it needs a (1) Mount Point and (2) File system information of the volume.

4. SSH into the EC2 instance
   chomod 400 keyname.pem
   ssh -i keyname.pem ubuntu@<public-ip>

5. Check to see the currently attached volume, can you see it? - No the attached volume is not visible.
   Check the vol data, it should be empty
   sudo file -s /dev/xvdf

6. Find the name of the block device from the shell. lsblk-command-in-linux.
   Check if EBS vol is in the disk
   sudo lsblk

7. Create a file system on the created EBS volume as per specifications. You need to create a file system on the new volume, don't modify any other file systems on any other volumes.
   Format the volume to ext3 file-syatem
   sudo mkfs -t ext3 /dev/xvdf

8. Create a directory on the host system(as per specs) and mount the file system-formatted volume to that directory.
   Create the folder
   sudo cd mnt
   sudo mkdir data-store

   Mount the folder onto the EBS vol
   sudo mount /mnt/data-store /dev/xvdf

9. List the current disks attached to the EC2 instance using the `df -h` command.
   sudo df -h

10. Snapshot the previously created volume:
    a. Give the description as: “Snapshot of my volume”
    b. Key: “Name”
    c. Value: “YOUR SRN_ebs”
11. Using the snapshot you just created, create another volume of 10 GiB. Name the newly created volume as “SNAPSHOT”

The specifications for the EBS volume are:

Volume Type	General Purpose SSD (gp2)
Size (GiB)	10
Availability Zone	Same availability zone as the EC2 instance
Encryption	Default(No Encryption)
File System	ext3
Mount Directory on host	/mnt/data-store

EC2 instance specs:

AMI	Ubuntu Server 18.04 LTS (HVM), SSD Volume Type
Instance Type	t2.micro
Instance Details	Default values
Storage	Default values

Note: Ensure the created EBS and EC2 volume belong to the same availability zone otherwise you will not be able to attach the volume

Creating an Amazon EBS volume - Amazon Elastic Compute Cloud

Creating & Attaching additional EBS

Viewing volumes and filesystems in Linux

Mounting volumes to directories in Linux

Task d : Object Storage using S3 Buckets

Before getting into AWS S3, it is important to understand what an object store is and how they are different from other types of cloud storage: What is object storage?

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, Internet of Things (IoT) devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9's) of durability and stores data for millions of applications for companies all around the world.

Introduction to Amazon S3

AWS Cloud Object Storage

This lab on AWS S3 will focus on two parts:

1. Creating an S3 bucket,adding a file,modifying access
2. Enable and use versioning for objects in S3.

Your sub-tasks for 1 are to:

1. Create an S3 bucket with a unique name as per specs.
    a. How do I create an S3 Bucket? - Amazon Simple Storage Service
2. Upload an image file(mentioned in specs) to the S3 bucket.
    a. Uploading an object to a bucket - Amazon Simple Storage Service
3. Get the public URL for the uploaded image and check access by pasting the URL into a browser.
    a. How do I see an overview of an object? - Amazon Simple Storage Service
4. Change access of the bucket to allow objects to have public access.
    a. Change bucket public access
5. Make the object *public* so as to allow public access. Check access again in a similar manner as sub-task 3.
    a. Grant public read access to some objects in my Amazon S3 bucket

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures.

Object Versioning - Amazon Simple Storage Service

Using versioning - Amazon Simple Storage Service

Your sub-tasks for 2 are to:

1. Use the same S3 bucket in the previous step.
2. Enable bucket versioning for the previously created bucket.
    a. How do I enable or suspend versioning for an S3 bucket? - Amazon Simple Storage Service
3. Download the sample text file(mentioned in specs)
4. Upload the sample text file to the bucket, enable public access for the object and check public access.

5. Now modify the sample text file as follows:
    a. Add a new line to the file mentioning “This is version 2 of the file”
    b. Add a new line saying “My SRN is “ <Your SRN here>.
    c. Do not change the file name, if you change the file name the bucket will treat it as a different file
6. Upload the modified text file to the S3 bucket.
7. Enter the same URL of the text file as you did before uploading the modified text file - you should see the updated text.
8. Go back to the bucket overview page and ensure that the *List Versions* toggle is on.
9. Under the sample text file uploaded, a new line appears showing the older version of the text file.
10. Try to access the older version of the text file.

S3 bucket specs:

Bucket Name	Your SRN, letters of your SRN must be in small case
Image file	new-report.png
Text file	sample-file.txt

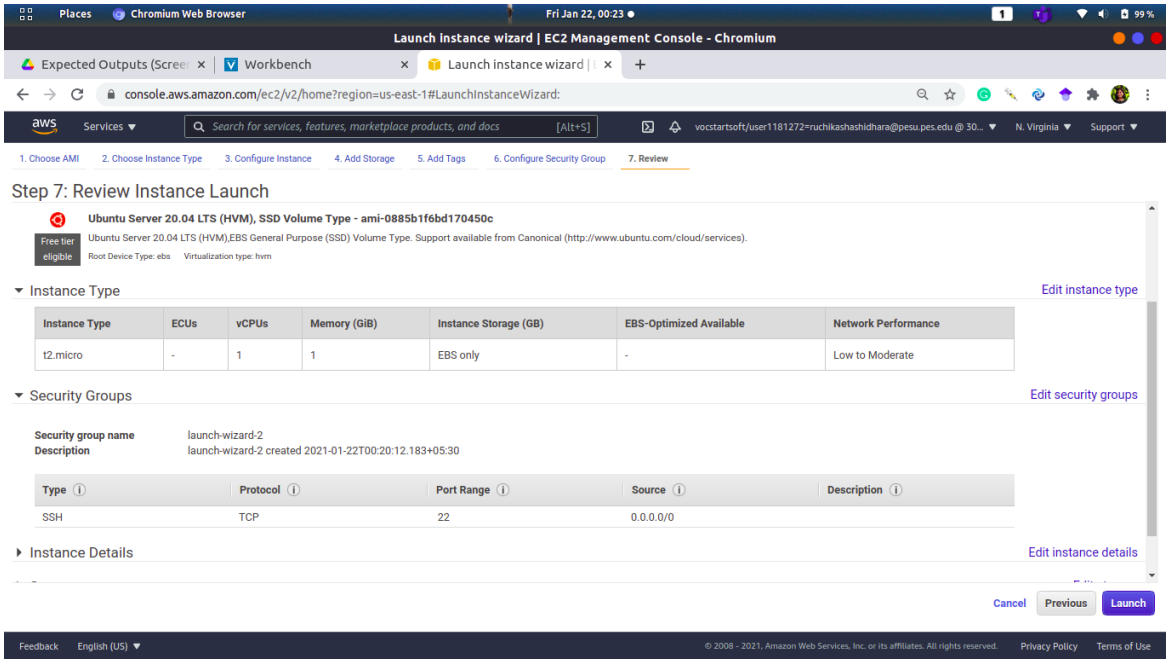
Few points to note:

1. Download the image and text file to your system before uploading to S3, don't change the file names while downloading.
2. Ensure the screenshots you upload clearly show the S3 object URL.

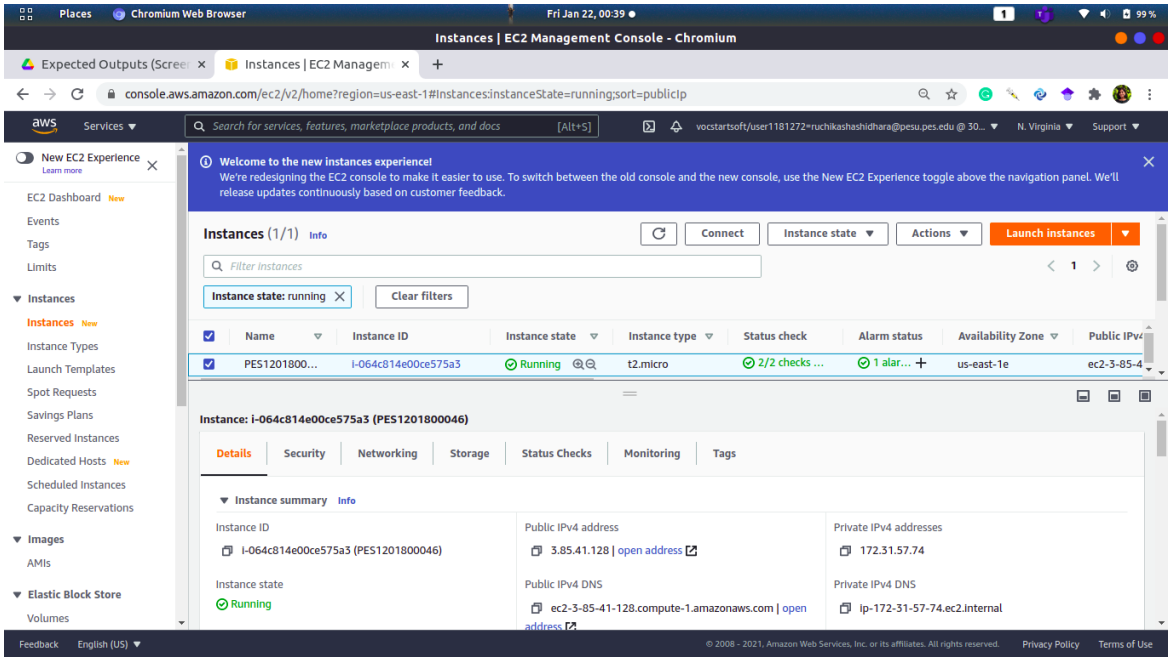
LAB 1 SCREENSHOTS

Task B:

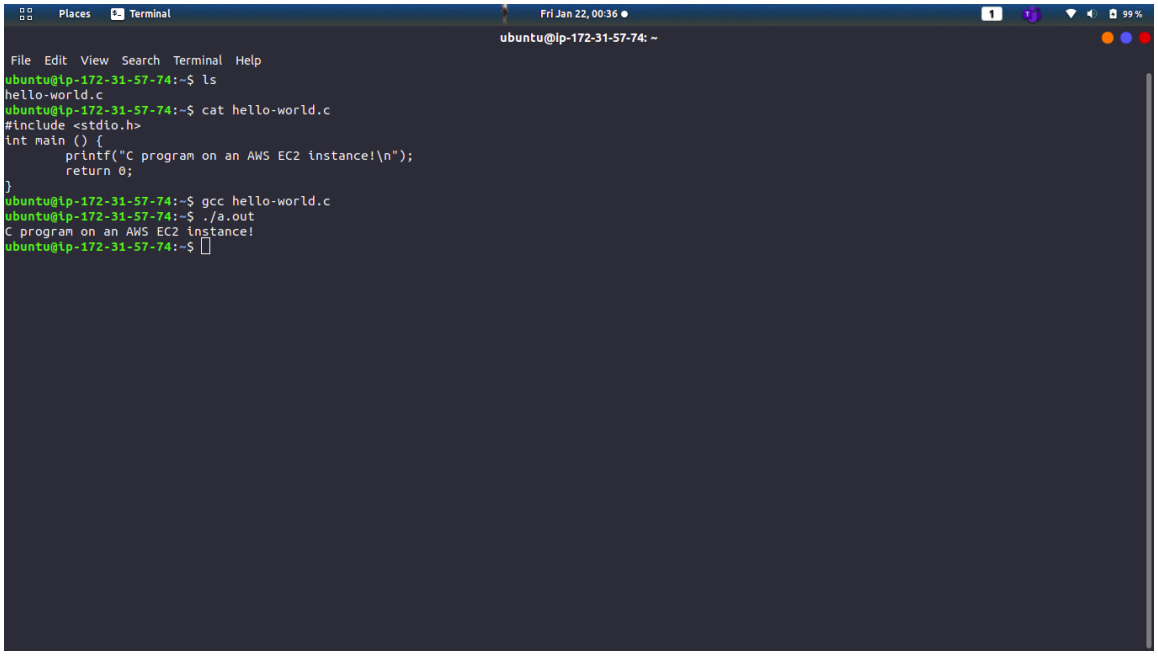
1b1.png showing the summary of the instance before launching (Step 7: Review Instance Launch)



1b2.png showing the EC2 instance in running state

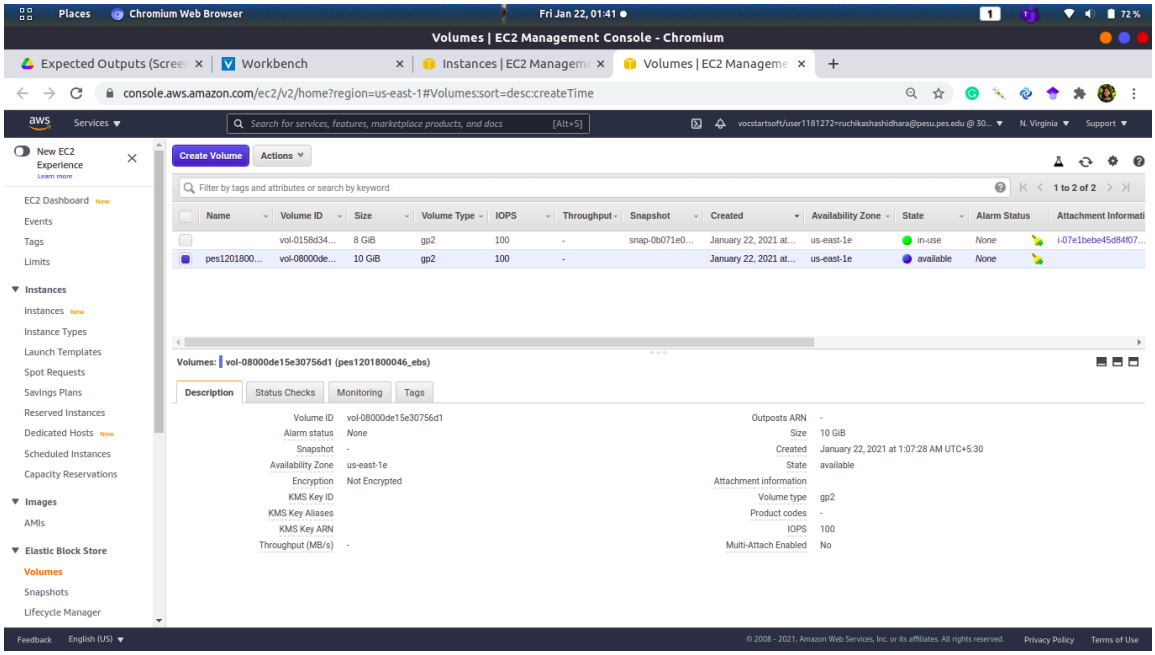


1b3.png, a screenshot of the SSH terminal, showing compiling and running a sample C program

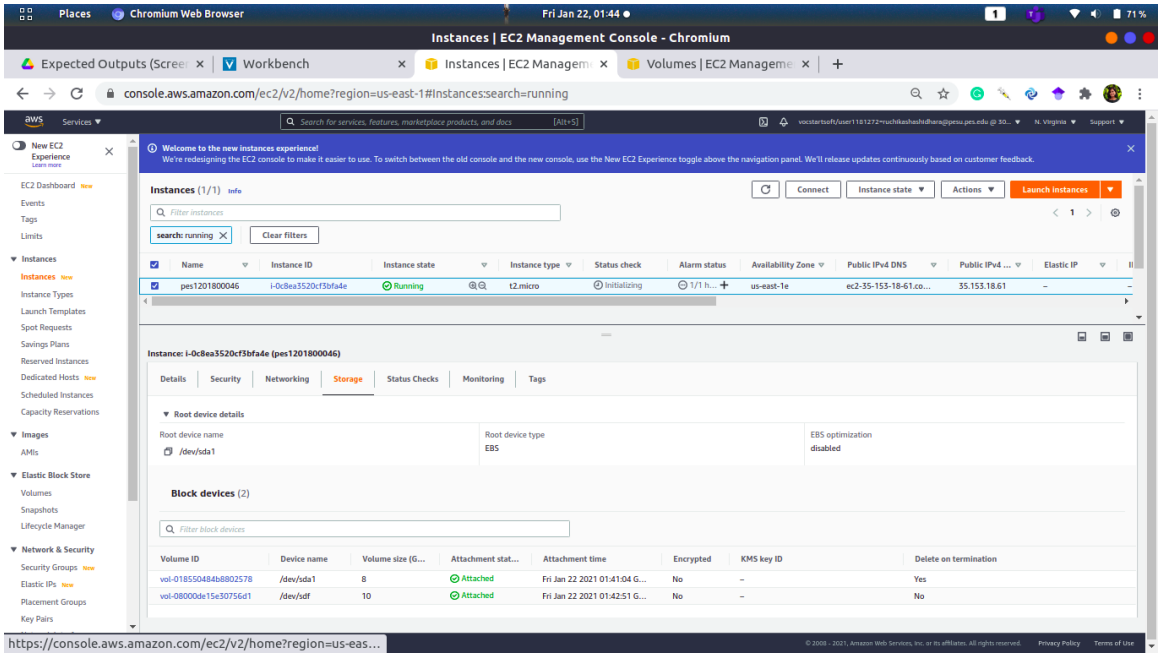


Task C:

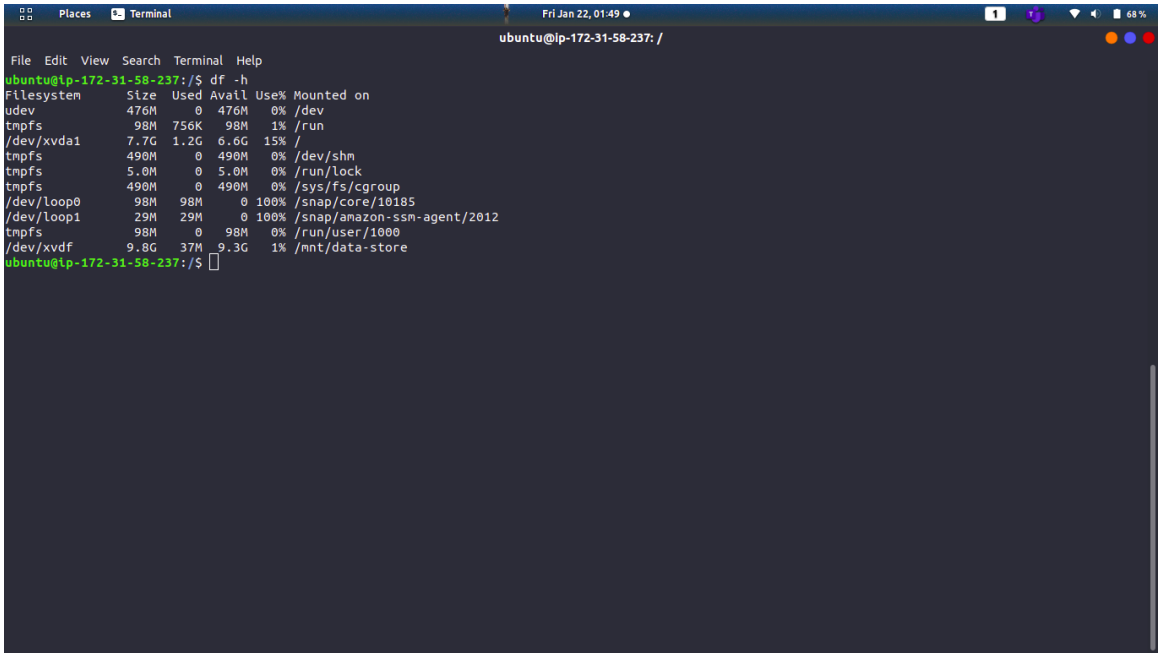
1c1.png showing the newly created EBS in an available state.



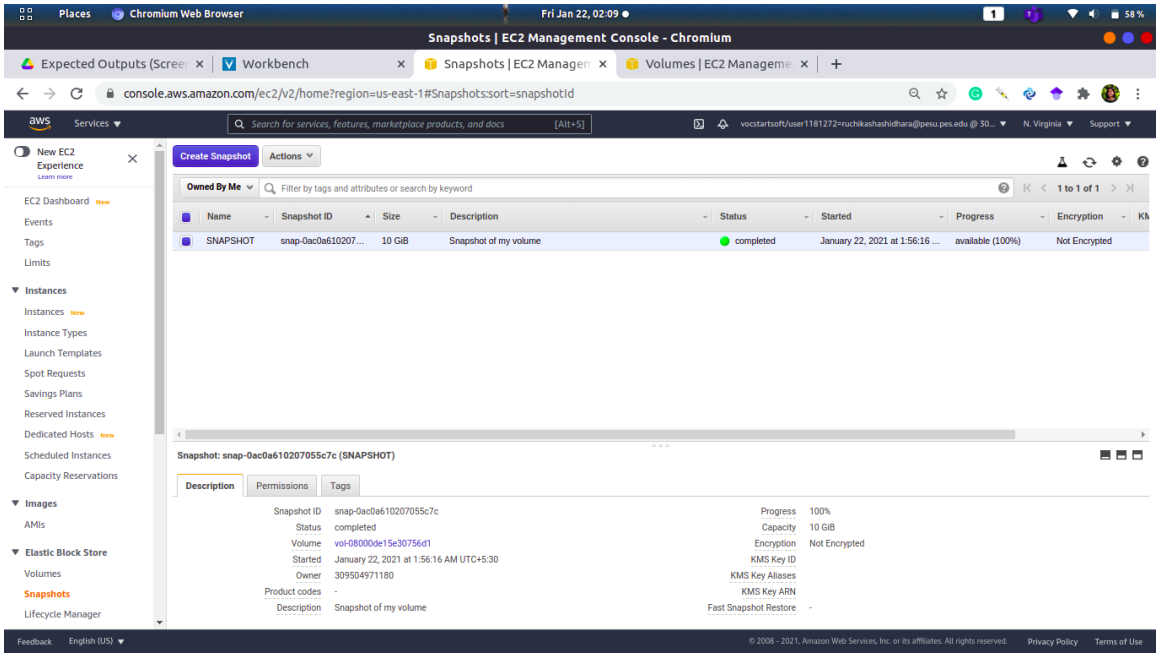
1c2.png showing the created EBS attached to an EC2 instance. The screenshot shows all the volumes of the EC2 instance on the EC2 management console.



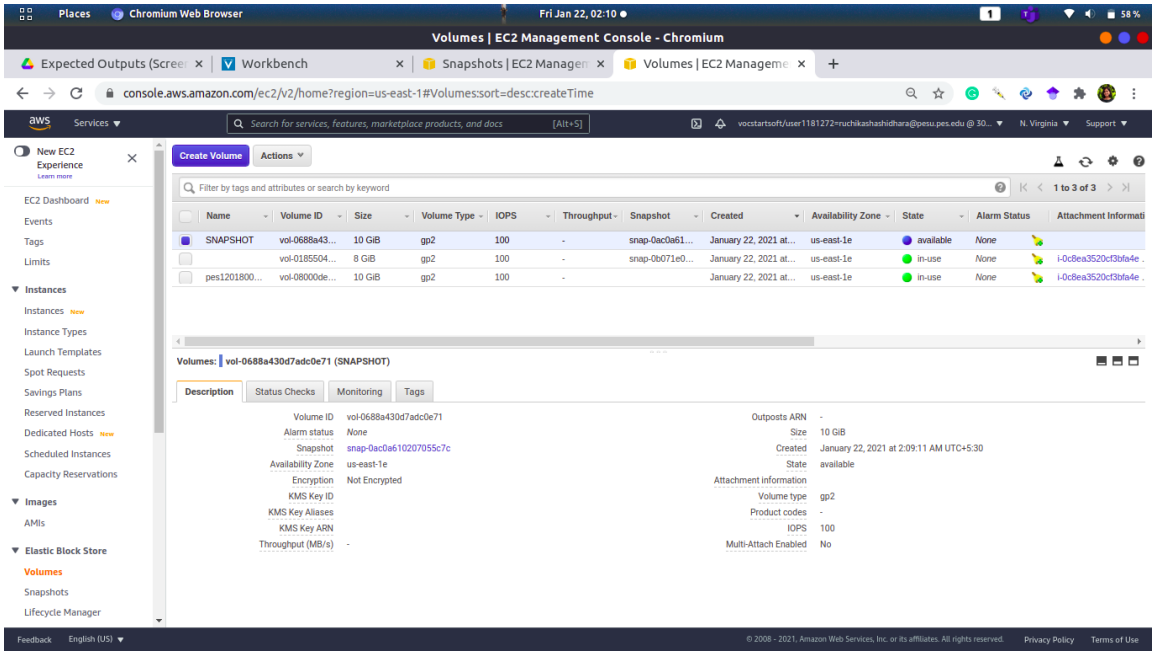
1c3.png showing an output of the terminal, by running a Linux command, showing all the volumes, file systems and size attached to the EC2 instance.



1c4.png showing the newly created snapshot on the management console

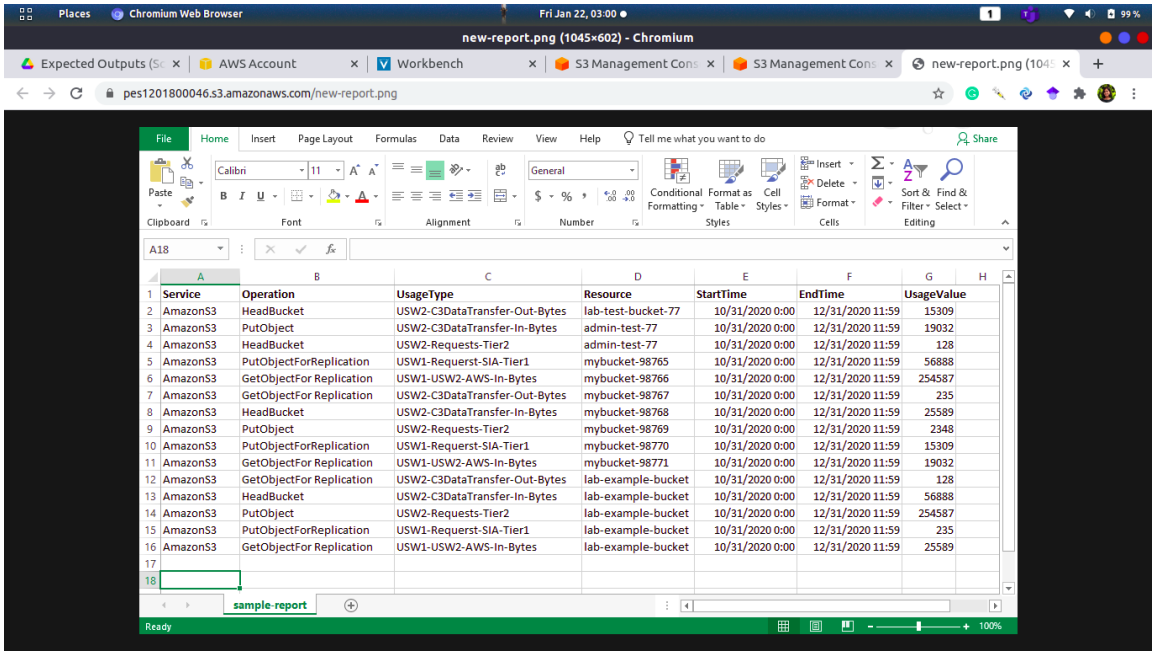


1c5.png showing the newly created volume from the created snapshot



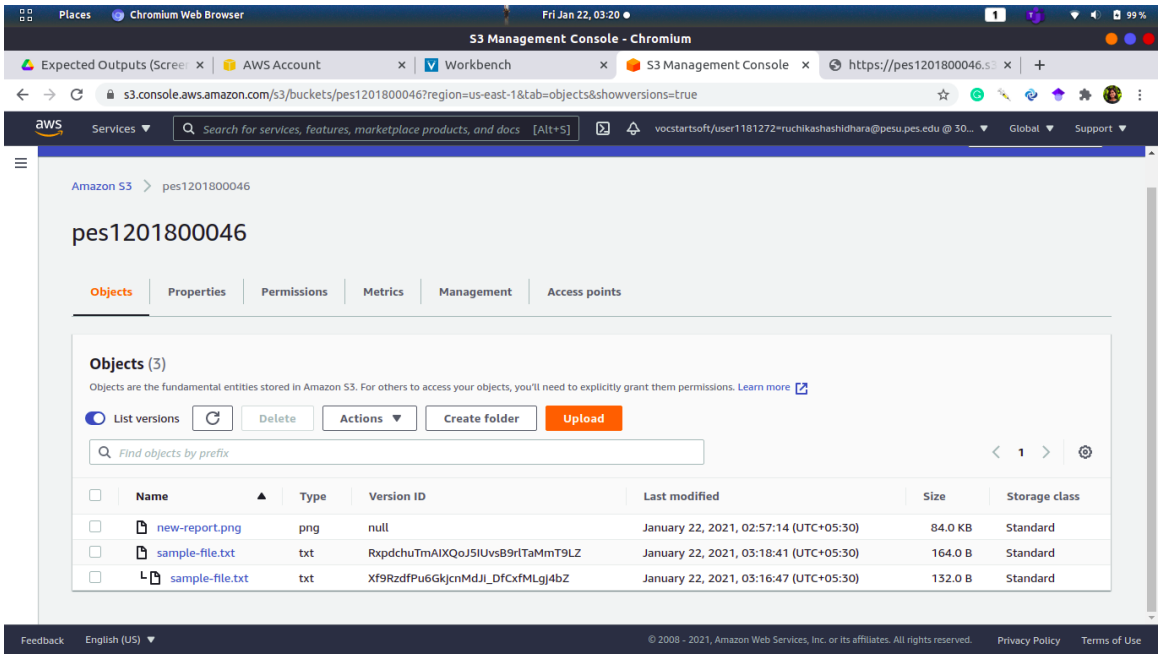
Task D:

1d1.png showing the uploaded image using the public URL

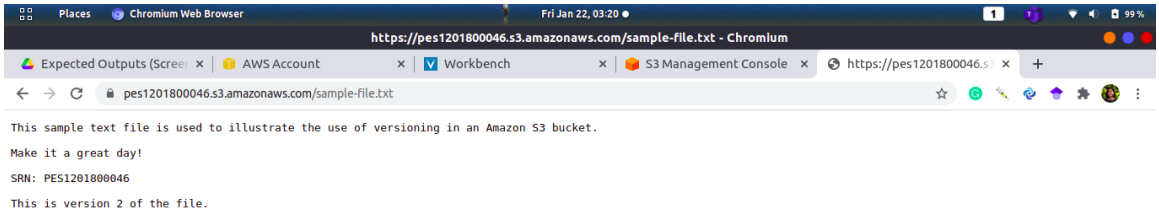


	Service	Operation	UsageType	Resource	StartTime	EndTime	UsageValue
2	AmazonS3	HeadBucket	USW2-C3DataTransfer-Out-Bytes	lab-test-bucket-77	10/31/2020 0:00	12/31/2020 11:59	15309
3	AmazonS3	PutObject	USW2-C3DataTransfer-In-Bytes	admin-test-77	10/31/2020 0:00	12/31/2020 11:59	19032
4	AmazonS3	HeadBucket	USW2-Requests-Tier2	admin-test-77	10/31/2020 0:00	12/31/2020 11:59	128
5	AmazonS3	PutObjectForReplication	USW1-Requerst-SIA-Tier1	mybucket-98765	10/31/2020 0:00	12/31/2020 11:59	56888
6	AmazonS3	GetObjectFor Replication	USW1-USW2-AWS-In-Bytes	mybucket-98766	10/31/2020 0:00	12/31/2020 11:59	254587
7	AmazonS3	GetObjectFor Replication	USW2-C3DataTransfer-Out-Bytes	mybucket-98767	10/31/2020 0:00	12/31/2020 11:59	235
8	AmazonS3	HeadBucket	USW2-C3DataTransfer-In-Bytes	mybucket-98768	10/31/2020 0:00	12/31/2020 11:59	25589
9	AmazonS3	PutObject	USW2-Requests-Tier2	mybucket-98769	10/31/2020 0:00	12/31/2020 11:59	2348
10	AmazonS3	PutObjectForReplication	USW1-Requerst-SIA-Tier1	mybucket-98770	10/31/2020 0:00	12/31/2020 11:59	15309
11	AmazonS3	GetObjectFor Replication	USW1-USW2-AWS-In-Bytes	mybucket-98771	10/31/2020 0:00	12/31/2020 11:59	19032
12	AmazonS3	GetObjectFor Replication	USW2-C3DataTransfer-Out-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	128
13	AmazonS3	HeadBucket	USW2-C3DataTransfer-In-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	56888
14	AmazonS3	PutObject	USW2-Requests-Tier2	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	254587
15	AmazonS3	PutObjectForReplication	USW1-Requerst-SIA-Tier1	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	235
16	AmazonS3	GetObjectFor Replication	USW1-USW2-AWS-In-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	25589
17							
18							

1d2.png showing different versions of the text file in the bucket overview



1d3.png showing the modified text file using the public URL of the file



Questions:

Task A:

Q1)The AWS management console is a web app to interact with AWS services. What other ways are there to interact with AWS services?

Ans) The other ways to interact with AWS services other than the AWS management console are:
i.AWS CLI - terminal management for quickness & repeatability
ii.AWS SDKs - interaction with code that we have written, helps with automation
iii. CloudFormation - tool that defines infrastructure as code

Task B:

Q1) What happens when you stop an instance? How is it different from terminating an instance?

Ans) When an instance is stopped, the data on the EBS volume will remain even after stopping but all information on the local (ephemeral) hard drive will be lost as usual. The volume will continue to persist in its availability zone. When an instance is stopped, we are not charged for any instance hours but only for the storage volume. When an instance is terminated, any storage is deleted and no charges are applied.

Q2) How are EC2 instances charged? How does it come under the pay-per-use model of cloud computing?

Ans) EC2 instances are charged based on size of the instance, OS or AWS Region where the instance is launched. It comes under the pay-per-use model as we can adapt our necessity depending on need and not on forecasts without overcommiting to budgets and swift response to changes.

Task C:

Q1) Is there any relationship between an Amazon AMI and the EBS? (Hint: Think about the default EBS attached to the volume, what all should it have configured?)

Ans) All AMIs are categorized by Amazon EBS, which means that the root device for an instance launched from the AMI is an Amazon EBS volume, or backed by instance store, which means that the root device for an instance launched from the AMI is an instance store volume created from a template stored in Amazon S3 during configuration.

Q2) What are the different types of EBS volumes available? What kind of volume is used as the boot volume? What should the different types be used for?
Ans) Provisional IOPS & General Purose SSD EBS Volmes can be used as the boot volume as it requires high I/O operations. Types of EBS Volumes:
i.Provisioned IOPS SSD (io2 and io1) - used in large databases and critical business applications for which the SSD volume is designed for mission-critical, low-latency & high-throughput applications.
ii.General Purpose SSD (gp3 and gp2) - Used in low-latency app, dev and test environments where the SSD volume provides a balance between performance & prices.
iii.Throughput Optimized HDD (st1) - Used in log processing, data warehouses, and streaming workloads for throughput-intensive workloads
iv.Cold HDD (sc1) - Used as cheap storage solution for infrequently accessed data.

Q3) How many volumes can you attach to a single EC2 instance? What factors does this depend on?

Ans) Yes we can attach multiple volumes onto a single EC2 instance. The factor it depends on is that the Volumes & Instance should be located in the same Availabiltiy Zone.

Q4) Can a single volume be bound to different EC2 instances?

Ans) No you can't attach a single volume to more than one instance as EBS provides a block storage abstraction of different filesystems such as ext3, ext4, etc.

Q5) What is the lifetime of an EBS volume? Is/Can an EBS volumes lifetime be bound to the lifetime the EC2 instance it is attached to?

Ans)

Task D:

Q1) This lab dealt with image and text files. What kind of objects can be uploaded to S3? Are there any types of objects you can't upload to S3?

Ans) We can upload any file type images, data, backups, etc. No there is no restriction of type of file, but in a single operation we can upload a single file upto 5GB in size, but if we use AWS CLI, we can upload a single file upto 160GB in size.

Q2) Is there any size limit for the S3 bucket? What is the largest size a single object can have in an S3 object?

Ans) Size limit of S3 bucket is 5 TB & size limit of a single object is 5 GB, but we increase the size limit using Multipart Upload Capacity.

Q3) Think of an application that uses S3 buckets to store user images. This application is used across continents(Americas,Europe,Asia). Should you use the same bucket across regions or multiple buckets across regions ? How would you do the latter?

Ans)

Lab 2 - Web Server, Firewall Access, Monitor Cloud Usage

Introduction:

Welcome to Lab 2! In this lab you'll learn how to set up a web server and host a web page. You'll learn how to Setup firewall rules. You'll also learn to monitor cloud usage.

Download files form here

Task A:

Objective:

The Objective of this task is to setup up a web server on the AWS EC2 Instance and host a sample web page on the instance. You are also required to load test the web page by using ApacheBench and monitor the EC2 instance using AWS CloudWatch service

- **Reading –** 20 mins
    - What is a Web Server? : Understand what a web server is.

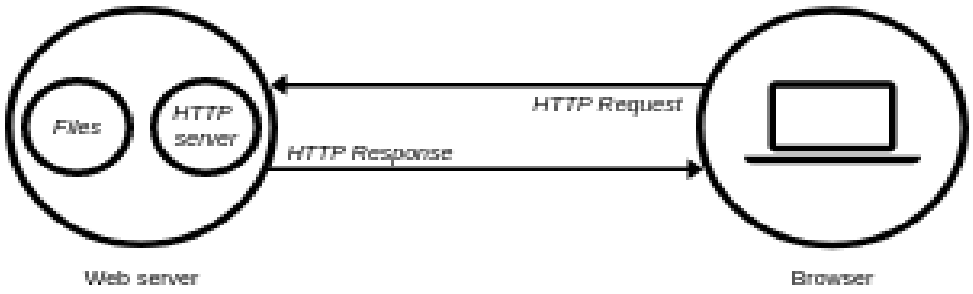
Sub Task 1: Launch a AWS EC2 Instance

- Launch a EC2 instance with the below Mentioned Configuration
- Follow **lab 1** steps to launch the instance.

EC2 instance specs:

AMI	Ubuntu Server 18.04 LTS (HVM), SSD Volume Type
Instance Type	t2.medium
Instance Details	Default values
Storage	Default values

Sub Task 2: install apache web server on the instance



It is important to understand the concept of web server before we jump into installing one.

The term web server can refer to hardware or software, or both of them working together.

- **On the hardware side**, a web server is a computer that stores web server software and a website's component files. (for example, HTML documents, images, CSS stylesheets, and JavaScript files) A web server connects to the Internet and supports physical data interchange with other devices connected to the web.
- **On the software side**, a web server includes several parts that control how web users access hosted files. At a minimum, this is an HTTP server. An HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages). An HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

Why Apache Web Servers?

Apache is considered open source software, which means the original source code is freely available for viewing and collaboration. One of the pros of Apache is its ability to handle large amounts of traffic with minimal configuration. It scales with ease and with its modular functionality at its core, you can configure Apache to do what you want, how you want it. You can also remove unwanted modules to make Apache more lightweight and efficient.

Steps:

- SSH to the instance
- install Apache2
- check the apache2 status

Execute:

chmod 400 keyname.pem
ssh -i -i keyname.pem ubuntu@3.85.41.128
sudo apt-get update
sudo apt install apache2
sudo systemctl status apache2

If apache2 is not running, start & enable the server

sudo systemctl start apache2
sudo systemctl enable apache2

Sub Task 3: Setup Http and Firewall access

Normally, Amazon computers only allow shell logins via ssh (port 22 access). If we want to run a Web service or something else, we need to give the outside world access to other network locations on the computer.

Steps:

- Find “Security Groups” in the lower pane of your instance’s information page, and click on “launch-wizard-N” hyperlink
- Edit the Inbound Rules
- Add a new rule: HTTP, 80, Source Anywhere and save them.
- Wait for few seconds and copy paste the Public DNS of your instance in your local browser. You should be able to see apache2 home page.

Refer this link for detailed explanation.

Sub Task 4: Host a Sample Web page

- You need to host a simple html Web Page in your instance using Apache Web Server and you should be able to access that webpage from your local web browser
- Create a Folder inside your “\var\www\html\” folder of your instance
- Name the folder by your SRN
- Create a simple html Page having your Name and SRN
- Access the Webpage from your local Browser using Your Instances PUBLIC DNS.

Execute:

sudo cd /var/www/html
sudo mkdir pes1201800046
sudo cd pes1201800046
sudo nano ruchika.html

Enter file contents:

```
<!DOCTYPE html>
<html>
<body>
<h1> PES1201800046 </h1>
<h2> Ruchika Shashidhara </h2>
</body>
</html>
```

Sub Task 5: Apache Benchmark and Cloudwatch

What is Apachebench?

ApacheBench is a single-threaded command line computer program for measuring the performance of HTTP web servers. Originally designed to test the Apache HTTP Server, it is generic enough to test any web server.

What is AWS Cloudwatch?

Amazon CloudWatch is a monitoring and management service that provides data and actionable insights for AWS, hybrid, and on-premises applications and infrastructure resources. With CloudWatch, you can collect and access all your performance and operational data in form of logs and metrics from a single platform.

Steps:

- In this task you are supposed to create load on your web server which you have setup on the ec2 Instance by using apachebench from your local computer.
- Then Monitor the load using Amazon cloudwatch(By enabling Detailed Monitoring)
- Identify the available metrics

- Vary the load using Apachebench,Following Load should be generated :
    - Users:10 Hits: 2000
    - Users: 20 Hits: 4000
    - Users: 30 Hits: 6000
    - Users: 40 Hits: 10000
- Identify the Metrics Whose values are changing and Take Screenshot of the Graphs.
- In order to monitor CPU Utilization Install stress-ng on your ec2 instance and then Create stress load of 60% and 80%. Take SS of CPU utilization curve.

Install stress-ng on EC2 instance
sudo apt-get update -y
sudo apt-get install -y stress-ng

Install apachebench on local system
sudo apt-get update -y
sudo apt-get install apache2-utils -y

Check num of cpus, nproc, it shows 2 for t2.medium, use -c 2 to apply stress on both cpus

Stress on EC2 instance before ab testing
Apply 60% stress, execute **stress-ng -l 60 -c 2**

AB Test for the following cases on the local machine
ab -k -c 10 -n 2000
http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html
ab -k -c 20 -n 4000
http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html
ab -k -c 30 -n 6000
http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html
ab -k -c 40 -n 10000
http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html

Apply 80% stress, execute **stress-ng -l 80 -c 2** and repeat the above ab test commands

Reference Link:

- Apachebencmark
- Cloudwatch your EC2 Instance (Look for enabling Detailed Monitoring for an existing EC2 Instance)

TASK B: Deploying Flask app On AWS Elastic Beanstalk

Objective:

In this task you'll be asked to create a AWS Elastic Beanstalk environment and deploy the flask application provided to you. You are required to configure a RDS along with your Beanstalk environment and create a few tables and databases.

What is AWS Elastic Beanstalk?

AWS Elastic Beanstalk is an orchestration service offered by Amazon Web Services for deploying applications which orchestrates various AWS services, including EC2, S3, Simple Notification Service, CloudWatch, autoscaling, and Elastic Load Balancers.

AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

Flask is a lightweight Web Server Gateway Interface WSGI web application framework that was created to make getting started easy and making it easy for new beginners. With the tendency to scale up to complex applications.

FLASK?

Flask has its foundation around Werkzeug and Jinja2 and has become one of the most popular Python web application frameworks.

As a developer in developing a web app in python, you may be using a framework to your advantage. A framework is a code storage that should help developers achieve the required result by making work easier, scalable, efficient and maintainable web applications by providing reusable code or extensions for common operations.

References:

1. Elastic Beanstalk – Deploy Web Applications
2. Introduction to AWS Elastic Beanstalk

Note: In this Experiment you'll be provided with two files

a) **requirements.txt** having the list of libraries required by the flask app.
b) **application.py** Flask application file having two API's.

Steps:

1. Launch a AWS Beanstalk environment.
    a. Choose sample application code.
    b. Setup Application and Environment name (Application name should be your SRN).
    c. Select the environment type to python 3.6

Reference for Step1.

2. Adding an Amazon RDS DB instance to your environment
    a. Setup Username(It should be your SRN) and Passwords
    b. Setup a MySQL Instance.
    c. After creating RDS, goto RDS instance info page
    d. Edit the securitygroup of your RDS Instance like you did in TASK A for EC2 Instance. **Edit the Inbound rules to allow traffic from Anywhere.**

Reference for Step2.

3. Write a Python Script to connect to RDS Instance Then you need to create a Database and Table in the RDS Instance.
    a. Create a DB of your Name Make changes in the application.py based on it.
    b. Create a table in that DB as follows:
        i.

```
CREATE TABLE New_Users (
    Personid int NOT NULL AUTO_INCREMENT,
    username varchar(255),
    password varchar(255),
    age INT,
    mobile_number varchar(10),
    PRIMARY KEY (Personid)
);
```

Use mysql instead of creating a python script:

Connect to database
mysql -u pes1201800046 -h aa1kvxsoyp67z8i.cc55ufnmcqdn.us-east-1.rds.amazonaws.com -p
Enter pes1201800046 when prompted for password

Create the database
mysql> **create database ruchika;**

Use the database
mysql> **use ruchika;**

Create the table
mysql> **CREATE TABLE New_Users (**
Personid int NOT NULL AUTO_INCREMENT,
username varchar(255),
password varchar(255),
age int,
mobie_number varchar(10),
PRIMARY KEY (Person-id)
) ;

4. Deploy the Flask App on Beanstalk.(Deploy using GUI). Watch this video to make your life simpler.

Make below changes in application.py

```
app.config['MYSQL_HOST'] =
'aae4ytox42dbyx.cx7ncna0qmke.us-east-1.rds.amazonaws.com'

app.config['MYSQL_USER'] = 'pes1201800046'

app.config['MYSQL_PASSWORD'] = 'pes1201800046'

app.config['MYSQL_DB']  = 'ruchika'
```

Create a zip, execute **zip demo.zip application.py requirements.txt**
Upload the demo.zip and sample application & deploy it

In application.py, make the following changes:

5. The Flask App has two REST API's One API is used to register Users And Another API is used to verify the registered user.
    i. API- Register User
        1. JSON Input

```
{
   "uname":"Ram",
   "pswd":"qwerty",
   "age":22,
   "mob": 8050569402
}
```

        2. Accepted Method: POST
    ii. API- Verify User

1. JSON Input

```
{
   "uname":"UserName",
   "pswd":"Password"
}
```

2. JSON Output (ID, Username, Password, Age, Mobile Number)

```
[
   6,
   "UserName",
   "Password",
   22,
   "8898734571"
]
```

3. Accepted Method: POST

6. Test the API's in the deployed Application Using POSTMAN. Create your own User and verify the user. SS needs to be taken. Reference for postman

LAB 2 SCREENSHOTS

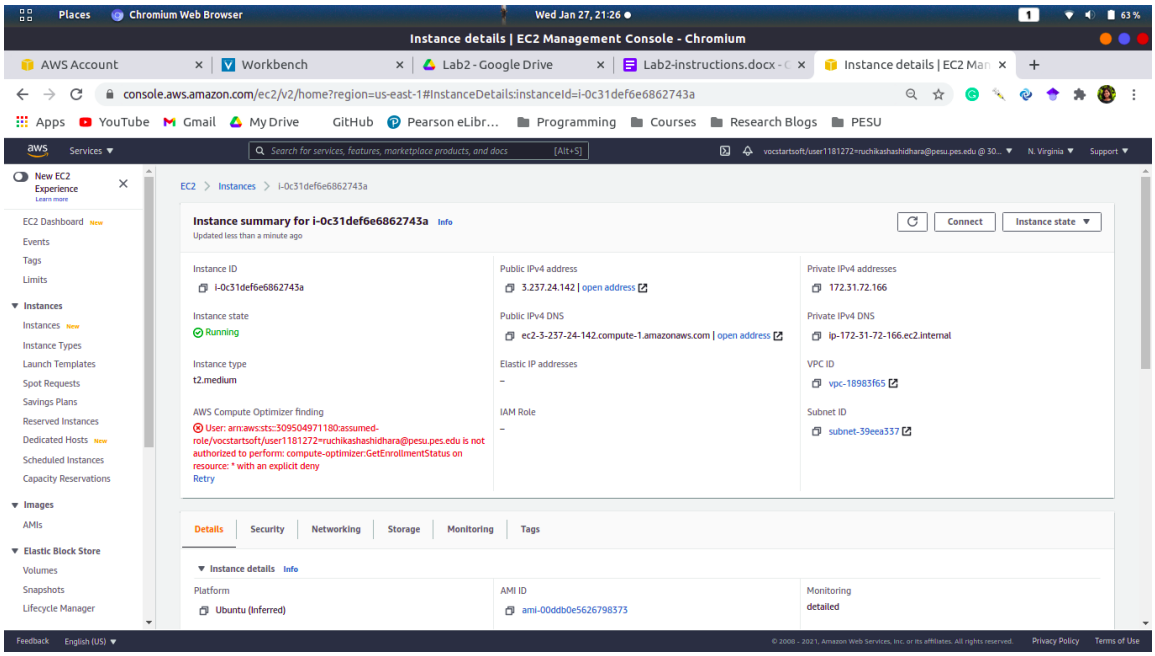
Task A:

Objectives:

1) Setup up a web server on the AWS EC2 Instance
2) Host a sample web page on the instance
3) Load test the web page by using ApacheBench
4) Monitor the EC2 instance using AWS CloudWatch service

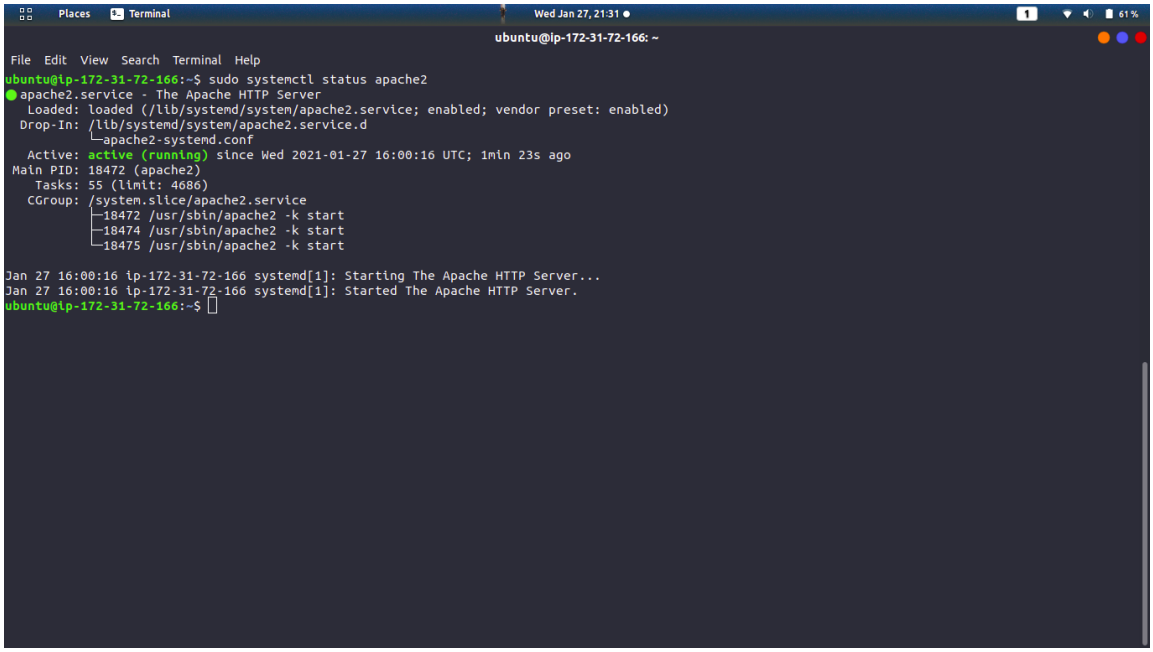
Sub Task 1: Create an EC2 Instance

1a.png showing that Instance is Running



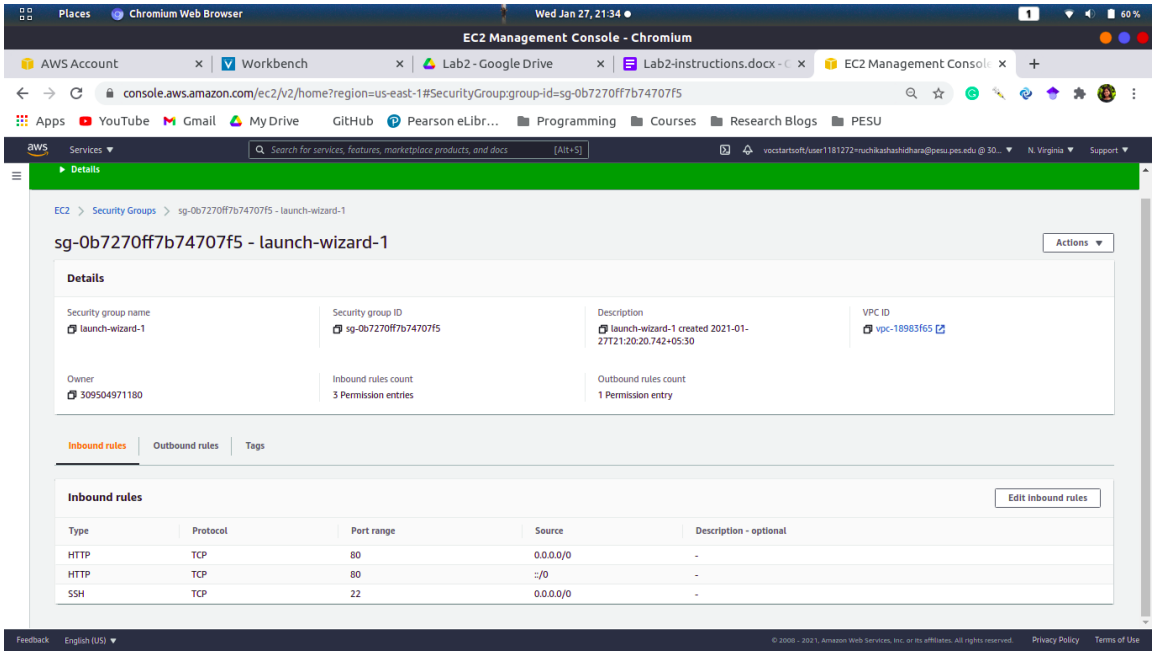
Sub Task 2: Install Apache web server on AWS EC2 Instance

2a.png showing that apache server status is active

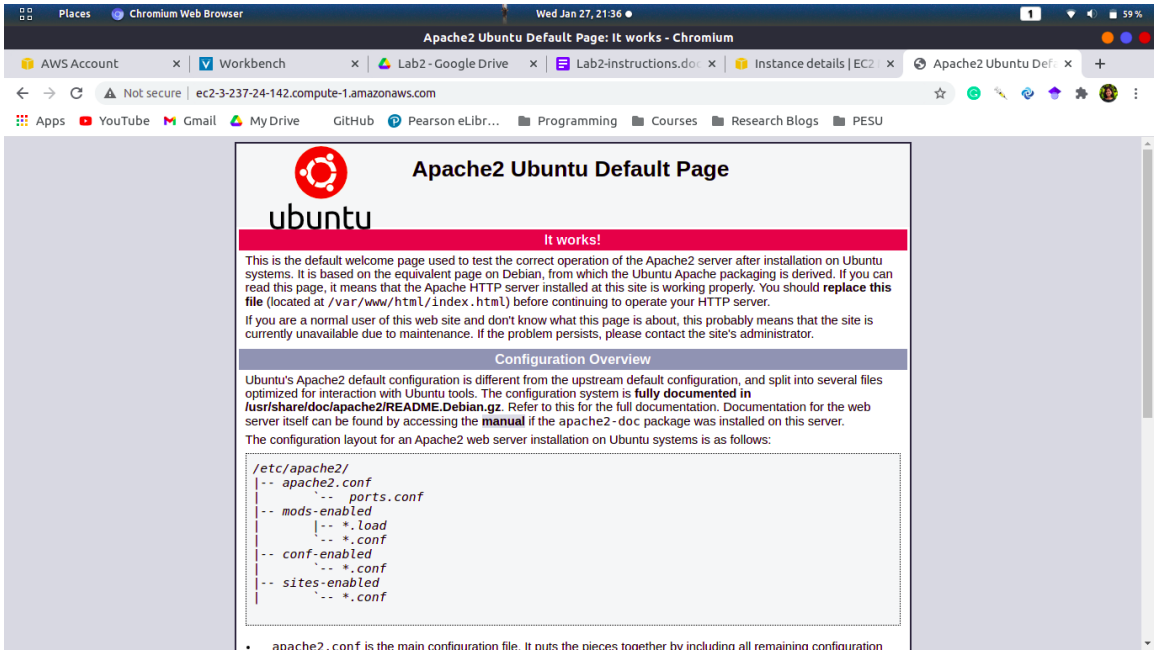


Sub Task 3: Setup Firewall Rules

3a.png showing the list of rules updated

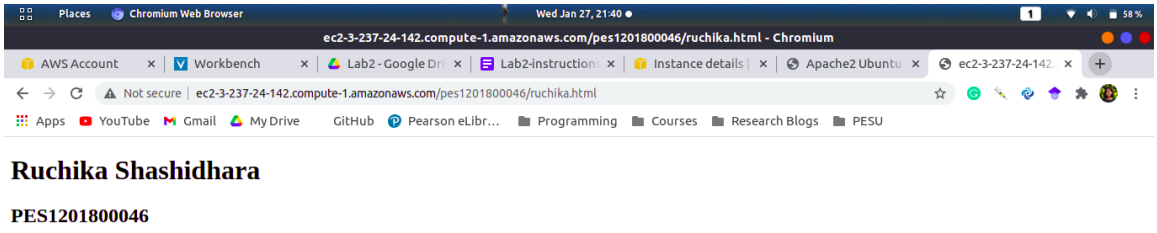


3b.png screenshot of the local browser showing EC2 Instance's Apache2 Ubuntu Default page using EC2's public DNS



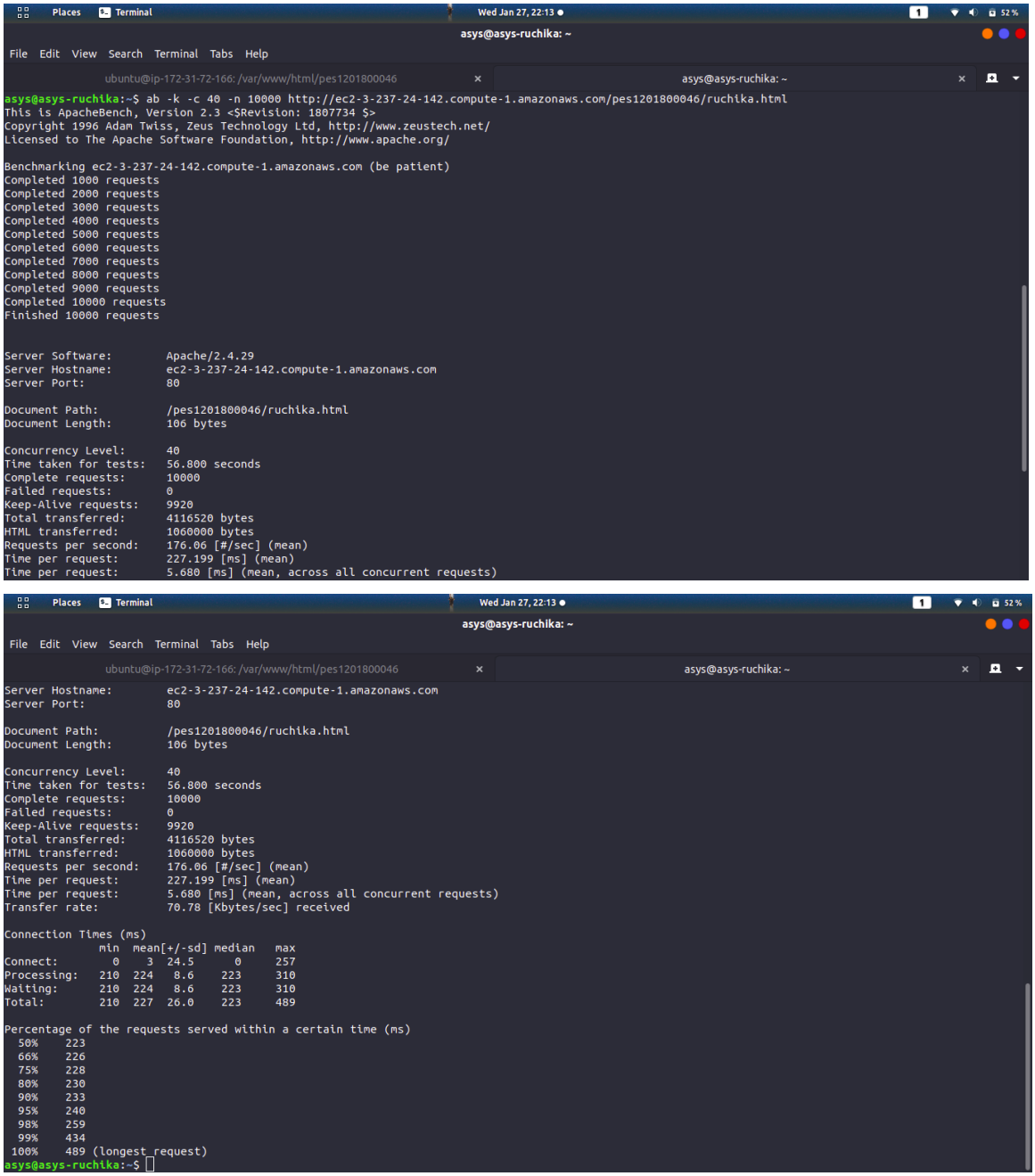
Sub Task 4: Host a Sample web page

4a.png screenshot of the hosted webpage accessed from the local computer

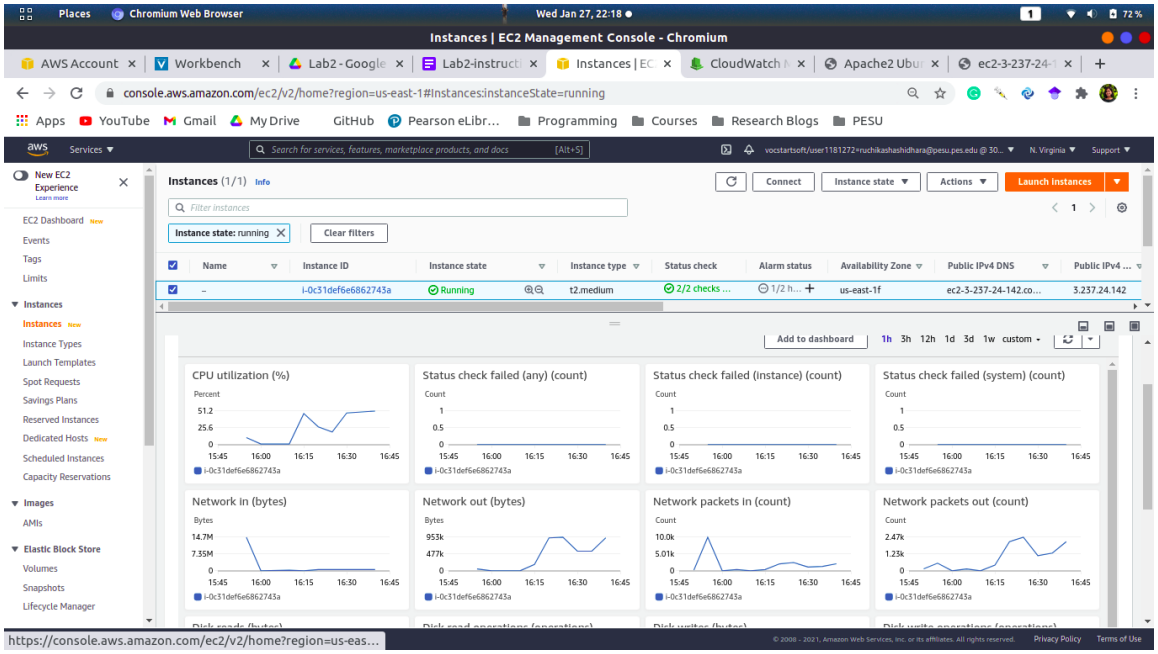


Sub Task 5: Apachebench and Cloudwatch

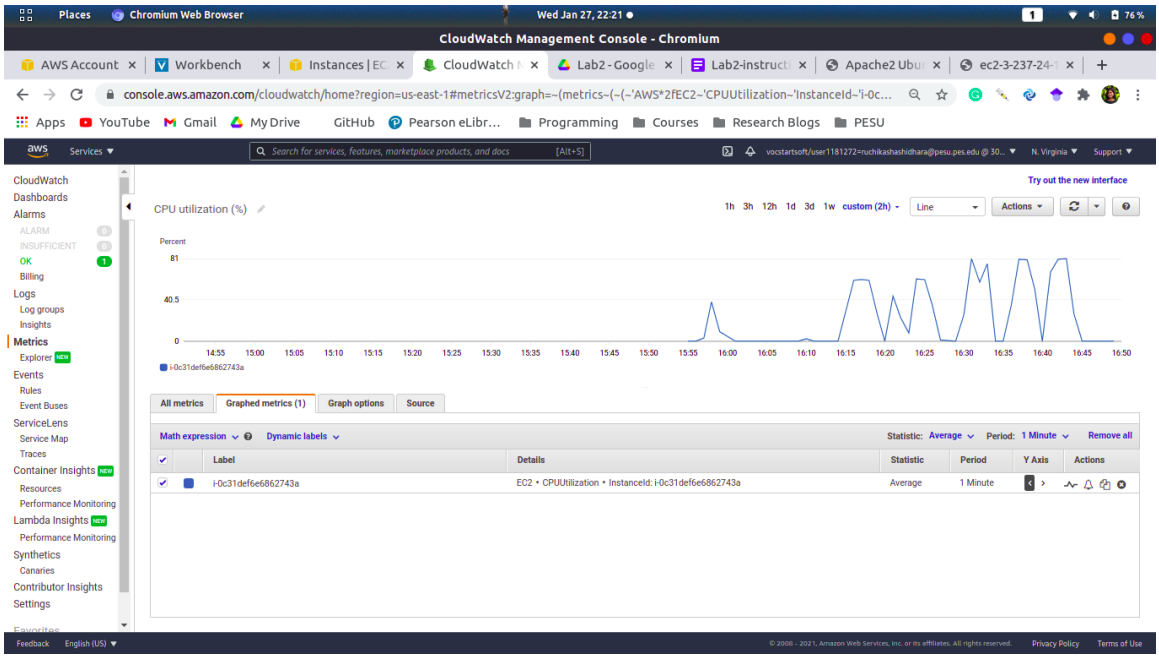
5a.png screenshot of the Apachebench output report after complete execution



5b.png screenshot of the all set of graphs



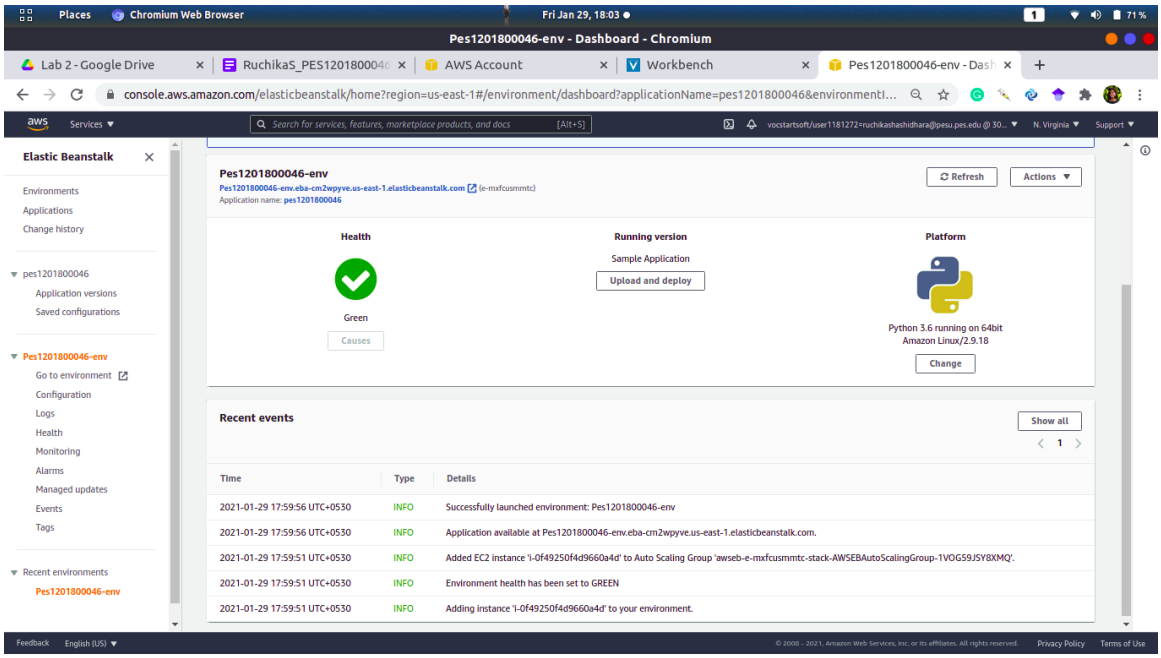
5c.png screenshot of CPU utilization



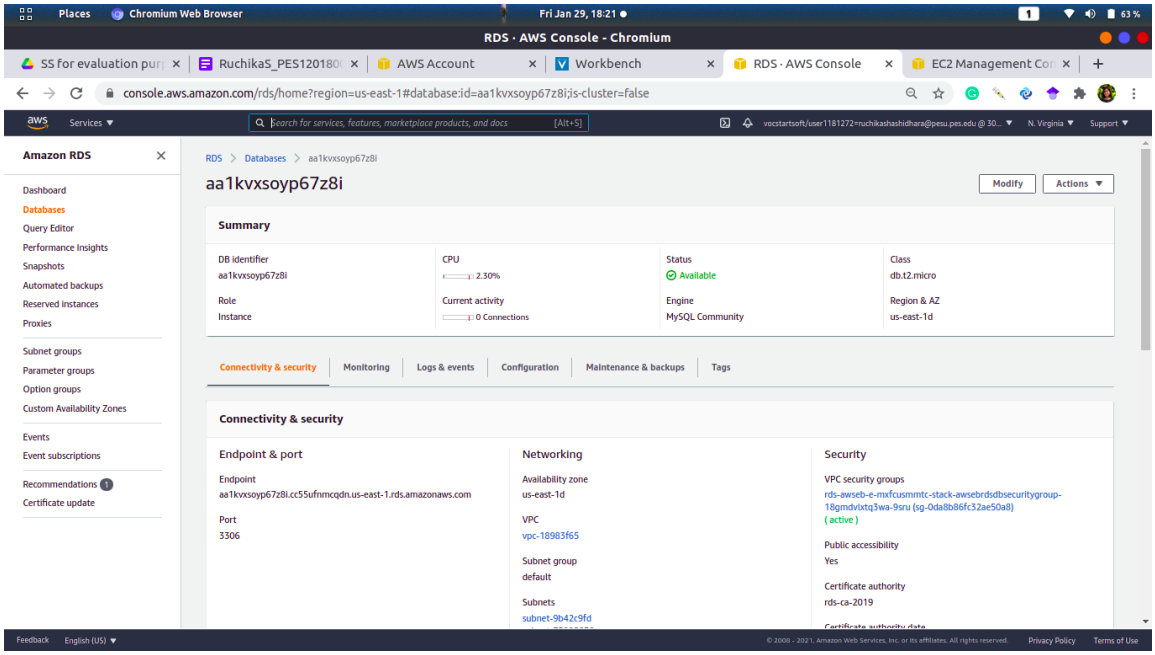
Task B:

Objective: Deploy Flask App on AWS Elastic Beanstalk

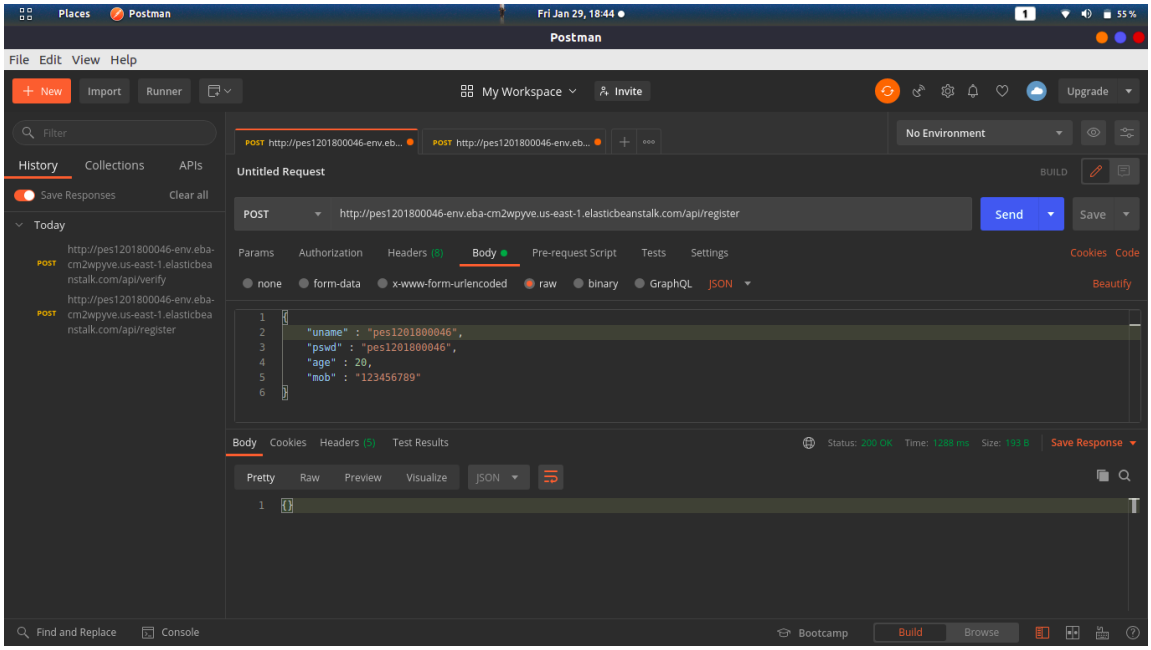
B1.png screenshot of Environment Health



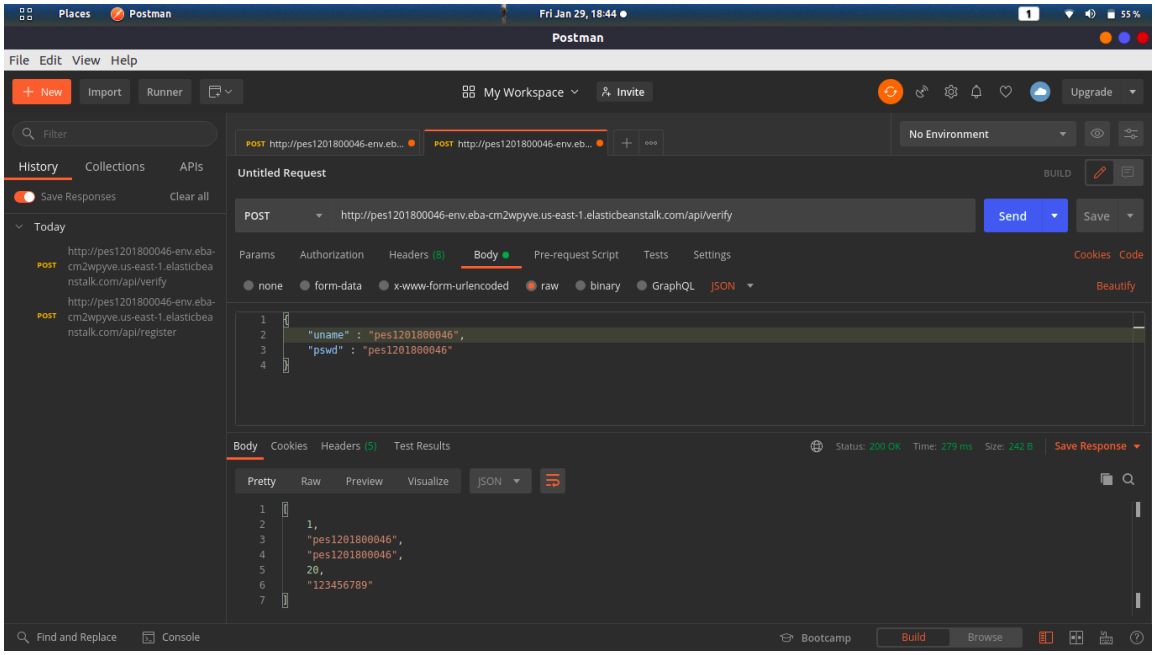
B2.png screenshot of DB configuration (RDS)



B3.png Register User Request (Postman screenshot)



B4.png Verify User Request (Postman screenshot)



Lab 3 - Virtual Private Cloud and migration of your existing application into the VPC.

Reading – 20 mins

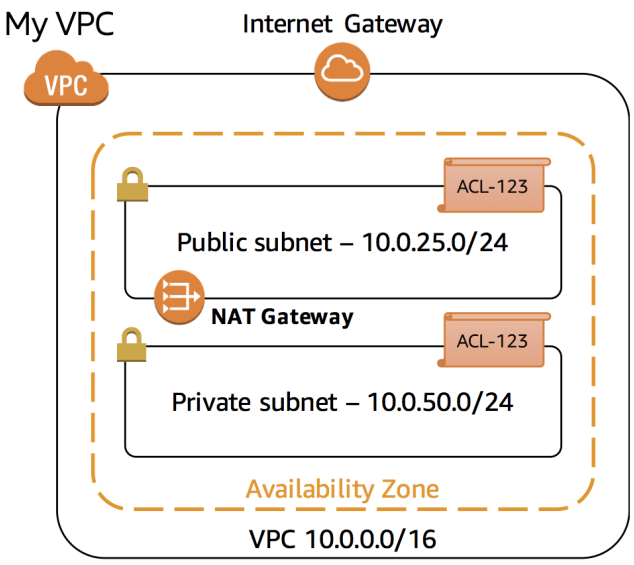
- Virtual Private Cloud : Make sure to go through benefits and use cases
- What is Amazon VPC? : Understand Key concepts for VPCs

Task 1: Create an Elastic IP address - -5 min

- Under services in the AWS management console, click on VPC
- Allocate an Elastic IP address with default settings.
- Take a screenshot with the allocated elastic ip address (3a.png)

Task 2: Create a VPC -35 mins

- Go to the VPC dashboard and launch the VPC wizard. The wizard makes your life much easier.
- Understand all the 4 VPC configurations
- Understand NAT gateways: NAT gateways - Amazon Virtual Private Cloud
- Create a **VPC with public and private subnets (Launch VPC wizard as shown in the demo video)**
- Configure the following:
  - VPC name: (your_name)_VPC
  - Public subnet IPv4 CIDR: 10.0.x.0 /24 (x: any number)
  - Private subnet IPv4 CIDR: 10.0.y.0/24 (y :any number)
  - Note: The availability zones should be the first in the list
  - Allocate the elastic IP address you created earlier.
- Create your VPC and click ok. Go to **Your VPCs** section and look at the details of the VPC you just created. Take a screenshot (3b.png)
- Internet gateway connects your VPC to the internet. Check if the gateway is attached to your VPC.
- Go to **subnets** and make sure to see if you have two subnets one public and one private.
- Take a screenshot of the subnets along with their ipv4 CIDR addresses. (3c.png)
- Go to the public network and take a screenshot(3d.png) and also analyse the details.



Task 3: Analyse your VPC -10 mins

- Each subnet is associated with a routing table which specifies the outbound traffic
- Routing tables for both your public and private networks
- The routing table for public network should have the following configuration:

- **Route 10.0.0.0/16** | **local** directs traffic destined for elsewhere in the VPC (which has a range of *10.0.0.0/16*) locally within the VPC. This traffic never leaves the VPC.
- **Route 0.0.0.0/0** | **igw-** directs all traffic to the Internet gateway.

- The routing table for your private network should look like this:

Route 10.0.0.0/16 | **local** directs traffic destined for elsewhere in the VPC (which has a range of *10.0.0.0/16*) locally within the VPC. This traffic never leaves the VPC.
Route 0.0.0.0/0 | **igw-** directs all traffic to the Internet gateway.

- Go to **Network ACL**:
- A network access control list (**ACL**) is an optional layer of security for your VPC that acts as a firewall for controlling traffic in and out of one or more subnets.
- Edit the inbound rule in your VPC to allow only http traffic Take a screen shot when done (3e.png)

Task b: -40 mins

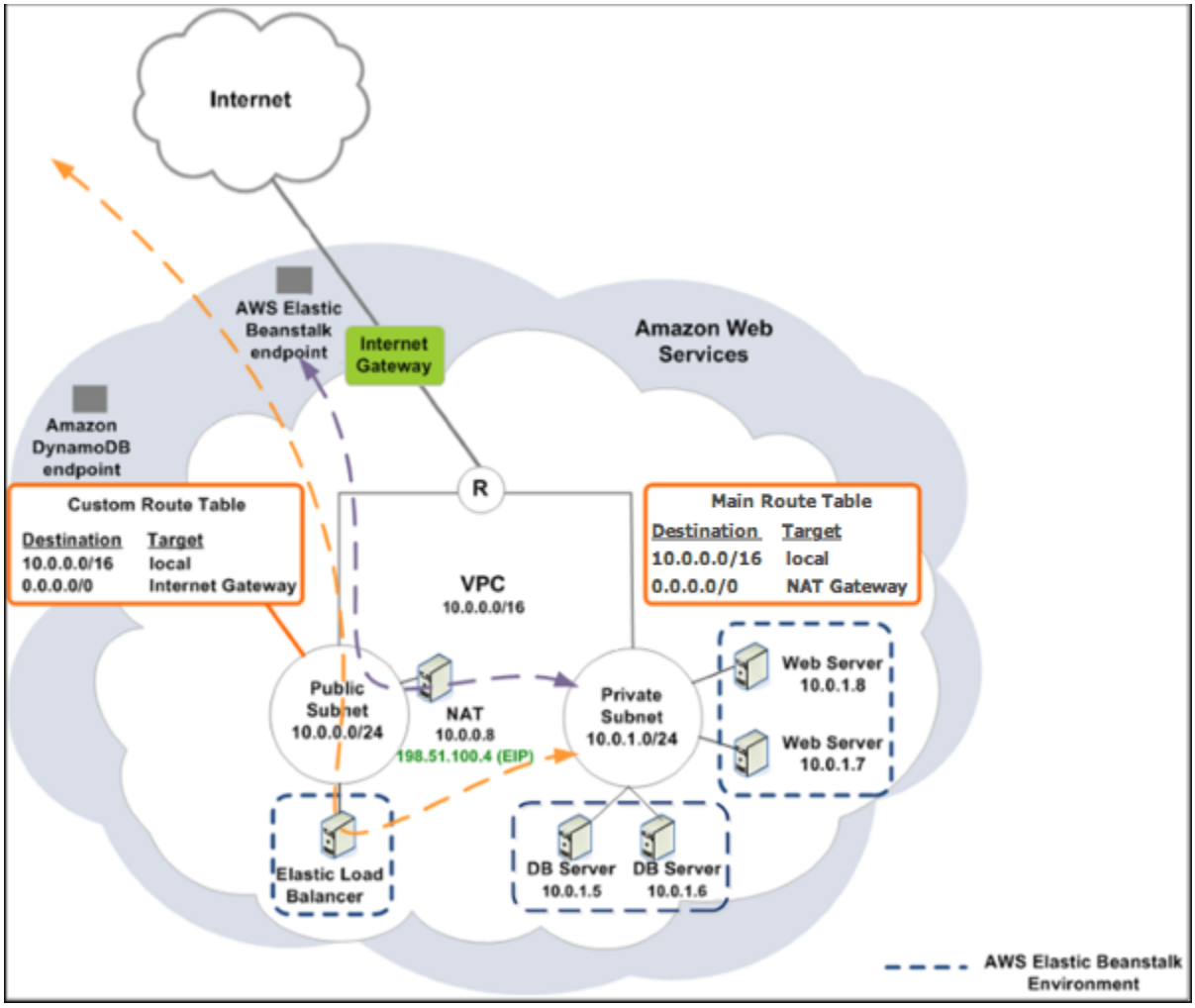
Migrating your existing application on Beanstalk into the VPC

(Note: Refer the pictorial representation below)

- Create a snapshot of your elastic beanstalk environment by saving your environment's configuration as an object in Amazon Simple Storage Service (Amazon S3)
- Make sure you have a snapshot of the RDS database which works along with your application.
- Refer this to create a saved configuration of your environment in beanstalk.
- Now, go to your saved configurations under your web app and choose **Launch environment.**
- Note: We are now going to launch this environment into our VPC created earlier.
- Give the new environment name: pes[yourSRN]-env
- Upload your application code by importing from S3 or locally.
- Click configure more options and modify network so that the app sits on the VPC you created.
- Assign your load balancer to a subnet in each Availability Zone (AZ) in which your application runs.
- Choose a subnet in each AZ for the instances that run your application. To avoid exposing your instances to the Internet, run your instances in private subnets and load balancer in public subnets.
- Now that you have your elastic beanstalk application in your VPC, take a screenshot (3f) clearly showing your environment name and the network in which your application resides.
- Make sure you can access your application through the URL provided in the beanstalk dashboard.
- Try editing your database (in configuration) and restore an existing snapshot of the RDS SQL database.
- Validate that your VPC and the beanstalk application if it's configured properly.
- You are Done! But don't forget to delete your beanstalk application and your VPC.

Task 4: Delete your VPC -10 mins

- Terminate all your beanstalk environments and the application.
- Delete the NAT gateway (only the one associated with your VPC)
- Detach the internet gateway associated with your VPC after deleting the private and public subnet attached with the VPC.
- Delete the VPC (all subnets and routing tables are deleted in this process)
- Note: Make sure deletion is done in order to prevent any dependency errors.
- Release the elastic IP that you allocated.



Reference: https://docs.amazonaws.cn/en_us/elasticbeanstalk/latest/dg/vpc-rds.html

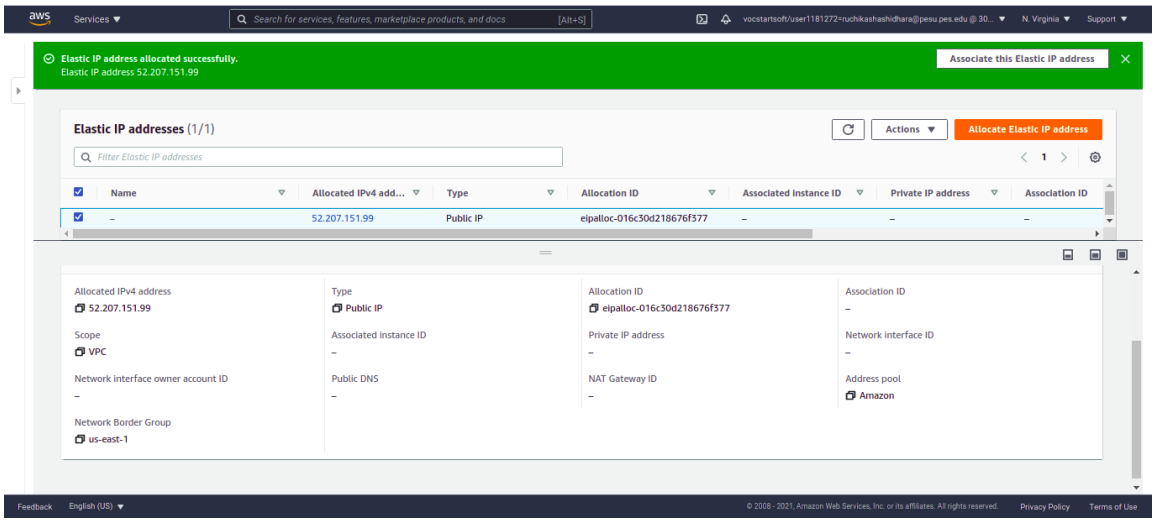
LAB 3 SCREENSHOTS

Outcomes of Task a:

- Create a VPC Network
- Create subnets across availability zones
- Understand connectivity within & between subnets
- Understand NAT, ACLs & Routing Tables

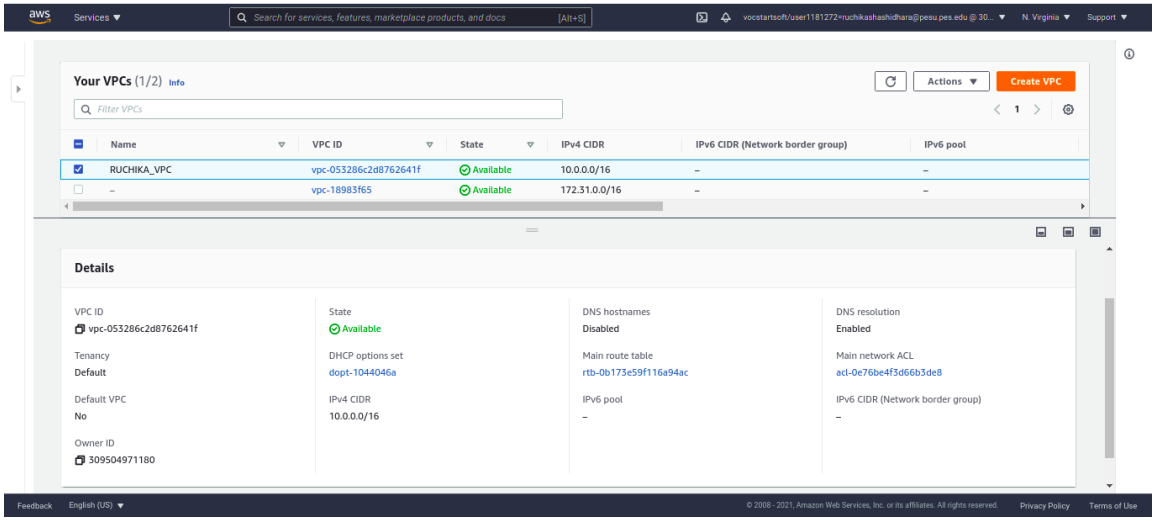
Task 1: Create an Elastic IP Address

3a.png - Allocated Elastic IP Address Details

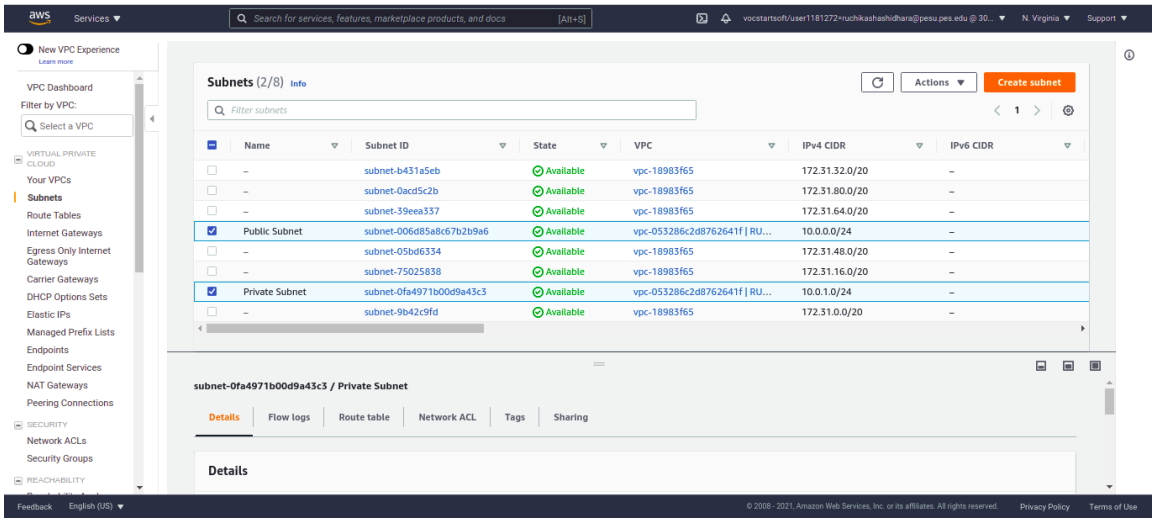


Task 2: Create VPC

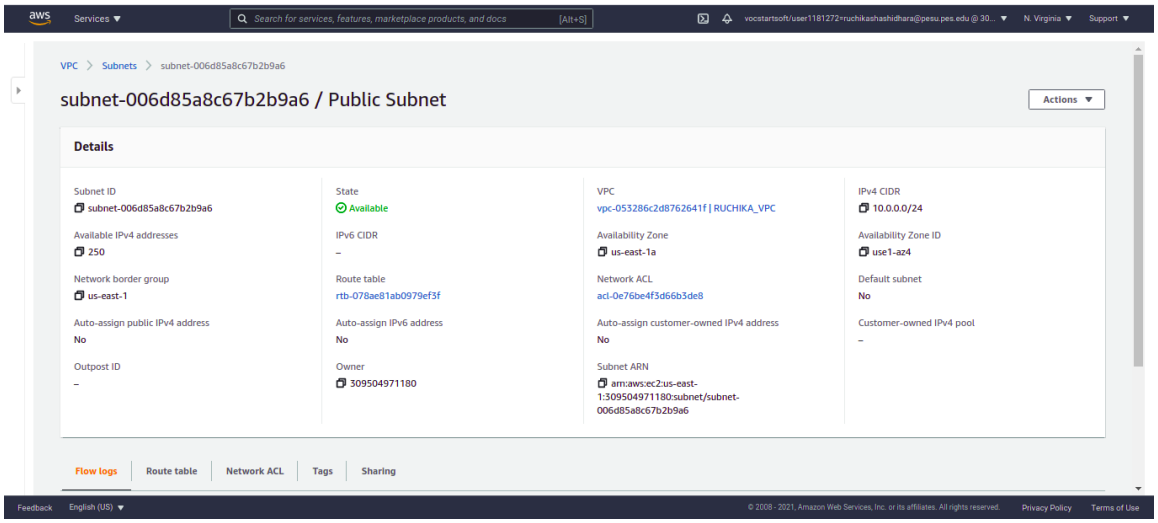
3b.png - VPC Details



3c.png - Subnets Details showing 1 private and 1 public subnets & their IPv4 CIDR addresses

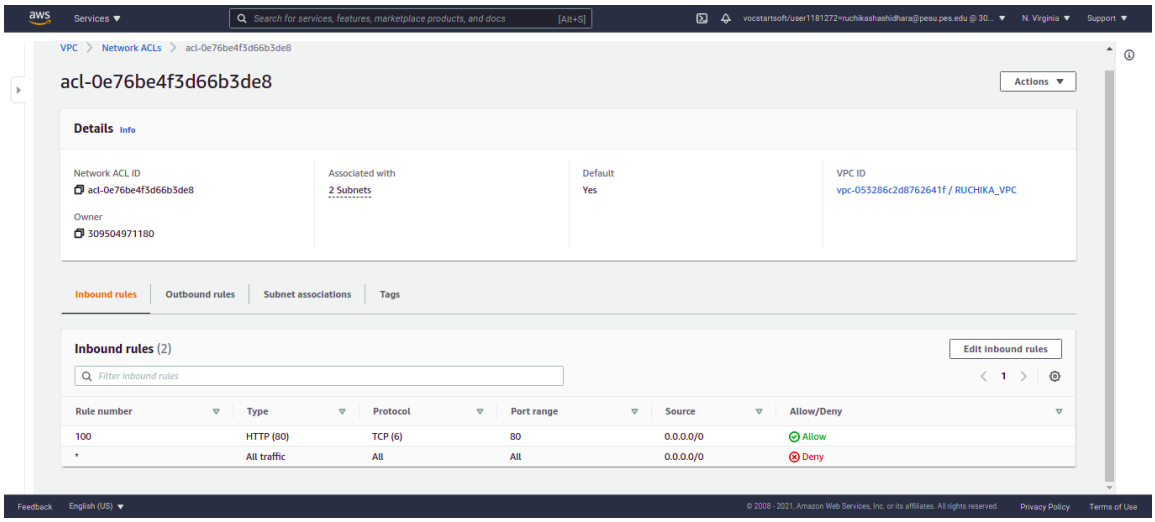


3d.png - Public Network Details



Task 3: Analyse the VPC

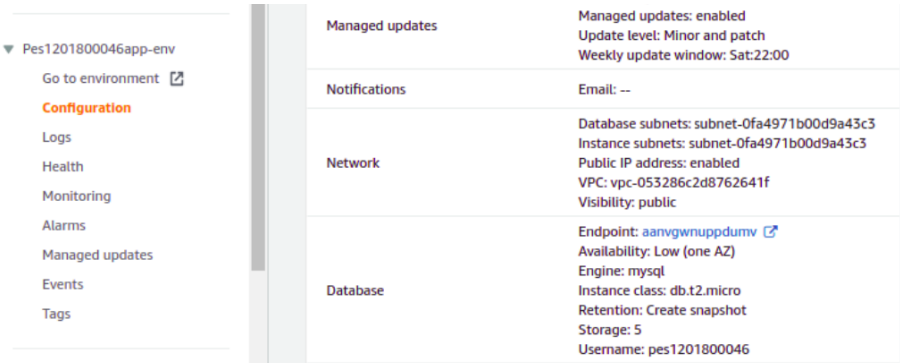
3e.png - Updated Inbound Rules Details showing the VPC to allow only HTTP Traffic



Outcomes of Task b:

- Migrate existing application on Elastic Beanstalk to the VPC
- Use Hybrid Cloud VPC i.e. both Public & Private Cloud
- Load the balancer in the Public Cloud & the application along with the Database that resides in the Private Cloud

3f.png - Migration of existing Elastic Beanstalk application into VPC showing Environment name & Network which the application resides on



Lab 4 - AWS SQS and SNS

Introduction to message queues for communication using AWS Simple Message Queue Service(SQS) and understanding publisher-subscriber messaging using AWS Simple Notification Service(SNS).

AWS SNS

- Amazon Simple Notification Service (Amazon SNS) is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication
- **SNS** is a distributed **publish-subscribe** system.
- Messages are **pushed** to subscribers as and when they are sent by publishers to SNS.
- Learn more here

Task 1 : Understand publisher subscriber pattern through application-to-person (A2P) messaging.

- Go to SNS from your console and create a topic
- Choose Standard as the type of your topic.
- Name the topic: [yourSRN]-[yourname] (eg: pes120180000-ram)
- Now, its time to create a subscription to the topic that you just created
- Create two subscriptions. Set the endpoint to your email and your friend's mail with the Email protocol. Make sure to confirm subscription from the mail inboxes. Once the subscriptions are confirmed, take a screen shot of all subscriptions. **(4a)**
- Now you need to publish messages to your topic.
- While publishing messages to your topic, set the header to "Hello this is [yourname] from SNS! " .
- Make sure to use Identical payload for all delivery, as your messages will be invariant of the different protocols.
- Send the message and check if you and your friend have received the mail. Take a screen shot of the mail received. **(4b)**

AWS SQS

- **SQS** is distributed **queuing** system.
- Messages are NOT pushed to receivers.
- Receivers have to **poll or pull** messages from **SQS**.
- Messages can't be received by multiple receivers at the same time. Any one receiver can receive a message, process and delete it.

- Other receivers do not receive the same message later.
- Polling inherently introduces some latency in message delivery in SQS unlike SNS where messages are immediately pushed to subscribers.
- Also note that SNS supports several end points such as email, SMS, http end point and SQS
- Learn in detail from here
- Learn about the basic architecture here
- SQS offers two types of message queues.:
    - Standard queues offer maximum throughput, best-effort ordering, and at-least-once delivery.
    - SQS FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.

Task 2 : Play around with SQS queues

- Make a standard queue and name it [yourname]-queue. Take a screenshot **(4c)**
- Send any message to the queue with a delay of 10 seconds.
- Poll for messages. Notice that the receive count gets incremented, if you poll for messages multiple times. Take a screenshot after polling multiple times. **(4d)**
- Unless you delete the messages, it remains in the queue.
- Delete the message from the queue.
- Send another message, and poll for messages.
- Try changing the queue to FIFO and experiment with it. Do you find any difference ?

Fanout to Amazon SQS queues

- Using Amazon SNS and Amazon SQS together, messages can be delivered to applications that require immediate notification of an event, and also persisted in an Amazon SQS queue for other applications to process at a later time.
- When you subscribe an Amazon SQS queue to an Amazon SNS topic, you can publish a message to the topic and Amazon SNS sends an Amazon SQS message to the subscribed queue.
- The Amazon SQS message contains the subject and message that were published to the topic along with metadata about the message in a JSON document.

Task 3 : Using Amazon SNS and Amazon SQS together

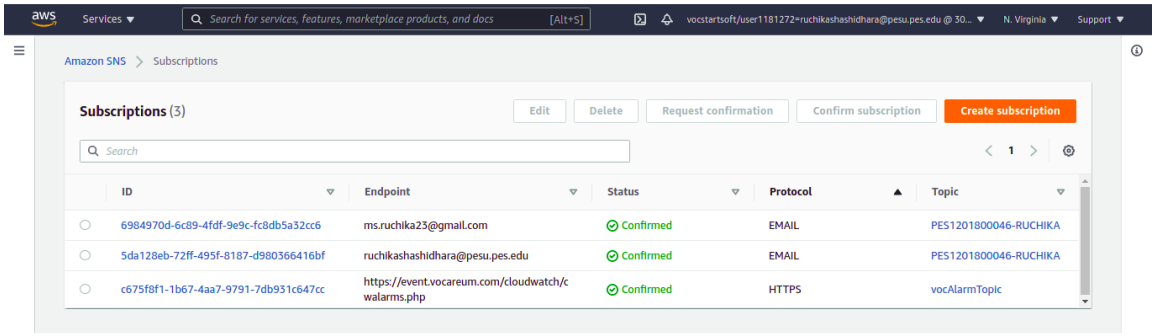
- Subscribe the queue which you created to the topic previously created.
- Publish a message to the topic and select custom payload for each delivery protocol.
- Go to your SQS queue and poll for messages. Take a screenshot of the json object which you received. **(4e)**

DELETE all topics, subscription and queues. You are done!!

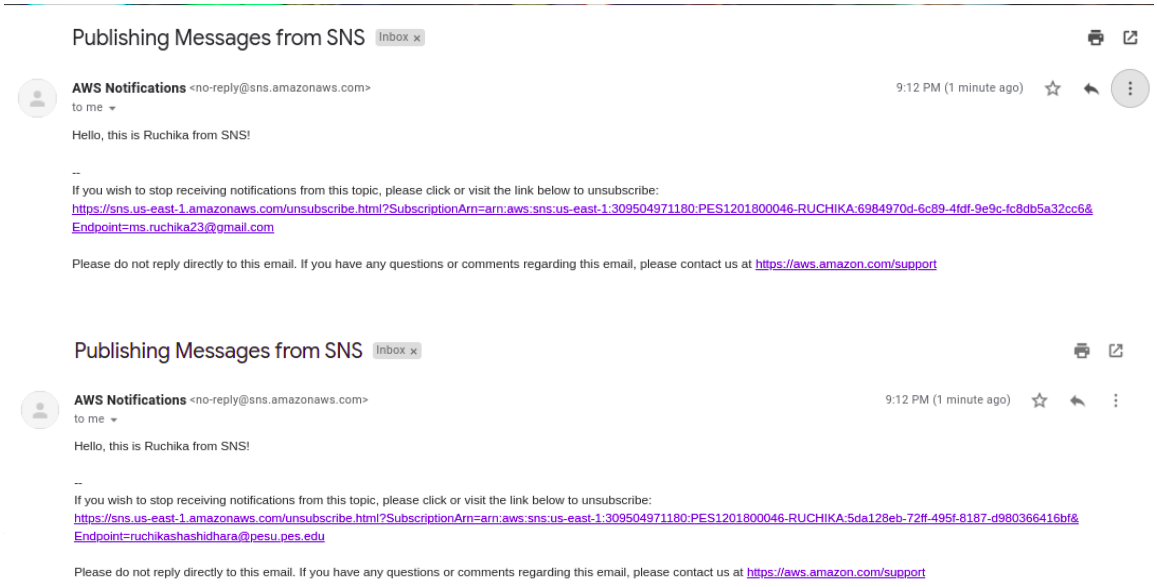
LAB 4 SCREENSHOTS

Task 1: Understand publisher subscriber pattern through application-to-person (A2P) messaging

4a.png - screenshot of all subscriptions

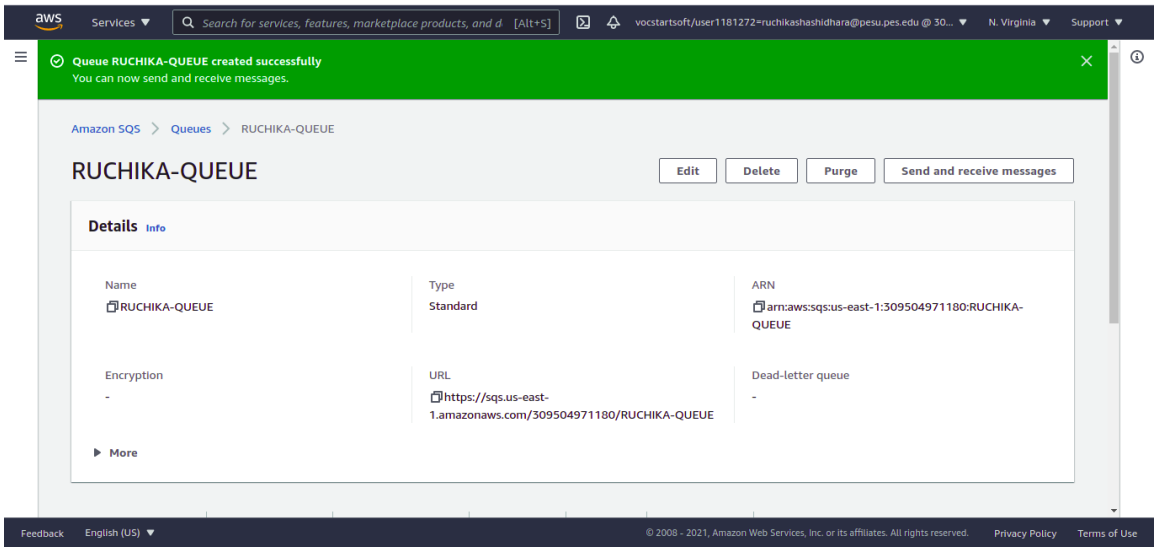


4b.png - SNS Messages - Mail Recieved

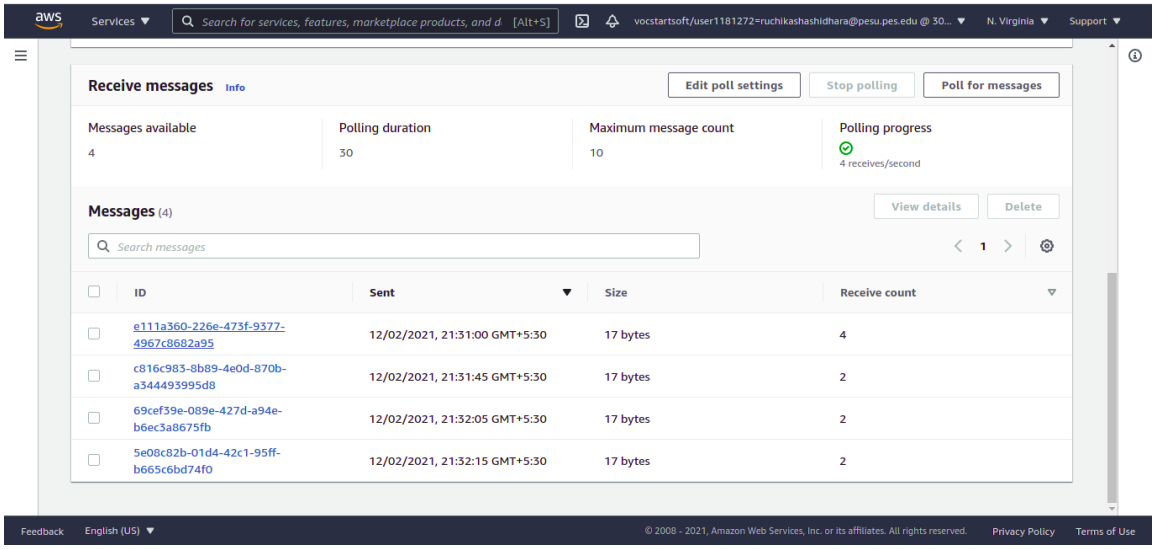


Task 2: Play around with SQS Queries

4c.png - SQS Standard Queue

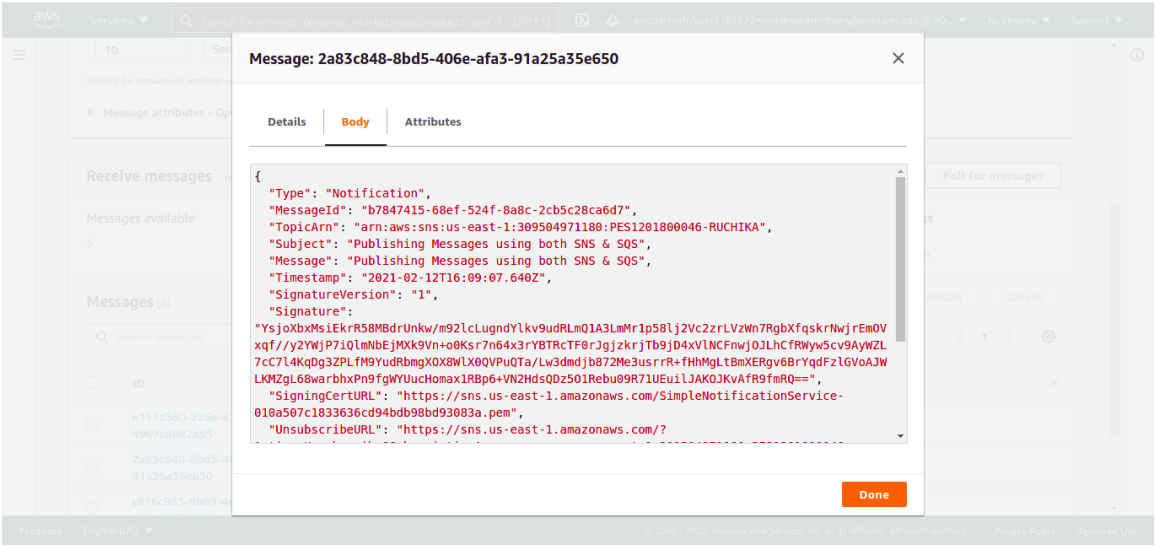


4d.png - Screenshot after polling the messages multiple times



Task 3: Using Amazon SNS & Amazon SQS together

4e.png - Screenshot of the received JSON object after polling the message from SQS



Questions:

Q1) Why do you generally couple SNS and SQS together?

Ans) We can couple SNS & SQS when we want to receive messages at our own pace. This allows us to deliver messages to applications to both applications which require immediate notifications of an event and persisted applications to process the messages at a later time. The latter allows the applications to be offline, tolerant to network & host failures and achieve guaranteed delivery of the messages later on.

Ex: If we use HTTP/Email endpoint in SNS, if there's a failure while sending messages due to heavy volume of messages, these messages are dropped and won't be received at the endpoints. But if we couple SNS & SQS, the same messages can be received at the endpoints and it allows a guaranteed delivery of the messages.

Q2) What's the point of having two different types of queues?

Ans) Both FIFO & Standard Queues can be used for different applications.
Standard queues can be used in scenarios when the applications can process messages that arrive more than once (allows atleast-once guaranteed delivery) and in an out of order manner.
FIFO queues can be used when we want to enhance the messaging between applications when the order of operations, events and message are crucial and requires no duplicates i.e. exactly-once processing mechanism.

Lab 5 - Docker

Few key points to note:

1. Apart from the attached resources, you can always refer to the official docker documentation. The docker documentation is well maintained and should help you through all your tasks.https://docs.docker.com/
2. Additional resources have been given at the end of the manual.
3. When you run docker commands in the foreground, you cannot access the command prompt in which case press **[Ctrl+C]** and continue with the next step or run docker in the background mode by using –d option as explained in the demo video.
   https://www.tecmint.com/run-docker-container-in-background-detached-mode/

Understanding containers and Docker

Please watch the below youtube tutorial **mandatorily** before starting the lab, this will help kickstart your understanding of Docker and the commands.

Docker Basic Commands | Docker Commands with Examples | Docker Commands Tutorial | Intellipaat

Containers are the de-facto standard for creating,maintaining and deploying microservices. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Container engines are the technology/software that are used to run the containers. There are many container engines available, but the most popular and easy to use CE is *Docker.*

The following links will help you get started on why we need containers, how they are different from VMs and why Docker:

What is a Container? | App Containerization - **Must Read!**

What is Docker?

Task 1: Installing Docker Engine on EC2

Sub-tasks are:

1. Create an EC2 instance with the following specs:

AMI	Ubuntu Server 20.04 LTS (HVM), SSD Volume Type
Instance Type	t2.medium
Instance Details	Default values
Storage	Default values

   Under configure security group, add rules HTTP and HTTPs - Allow Anywhere and SSH - Allow Anywhere (SSH will already be present)

2. Get keypair(pem) file and SSH into the instance.
3. Install docker engine on the instance , instructions are in the given link:
   a. Install Docker Engine on Ubuntu
4. Use the following link to make docker a “sudo-less” command:
   a. Post-installation steps for Linux
5. Verify that Docker Engine is installed correctly by running : docker run hello-world

After ssh, run the following commands for installation:

sudo apt-get update

**sudo apt-get install **
**apt-transport-https **
**ca-certificates **
**curl **
**gnupg-agent **
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

(Check if last 8 digits under docker shows: 0EBF CD88 in the command)
apt-key list or **apt-key finger**

sudo apt-key fingerprint 0EBFCD88

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker

docker run hello-world
```

Task 2: Docker images and docker files

In AWS EC2 instances we saw *AMI(Amazon Machine Images)*, which was basically the operating system and in case of snapshots, the current *state* of the VM. Docker images are similar, but are not the same. Docker images are “ready to go” meaning everything that needs to be installed(requirements) has already been taken care of when building the image and they don't need to be installed/taken care of again.

Docker files are used to create Docker images. Dockerfile are a series of steps that specify which base image to use, which files/folders to copy into the container, run which commands while starting up the container and which will be the main process of the container. We need to create dockerfiles, then build it into an actual runnable docker image.

Sub tasks:

1. Docker hub is a central public repository containing Docker images and documentation on how to use these base images. Explore the different images at : https://hub.docker.com/
2. The images in docker hub, need to be *pulled* into our EC2 instance in order to use them(Pulling is the equivalent of downloading the images onto you docker host).
    a. docker pull

Pull the following images onto your docker instance: Ubuntu 18.04, Nginx, Python, MongoDB. Stick to the latest images. To find the images that exist on your instance you can run the docker images command. This shows the image name, when you pulled the image, the tags(versions) of the images.

3. Now that we have the images we need to actually run the containers,
    a. We can run a container using the docker run command.
    b. To list the *currently running* container we use the docker ps command
    c. To stop a running container safely we use docker stop
    d. Usually after a container is stopped it goes into a "exited state", this can block some I/O operations needed by future containers and you will not be able to use the name of the exited container. What are the possible states for a docker container?

       To safely and cleanly remove a container, use the docker rm command.

4. *Containers are light weight VMs*, this means we should be able to somehow interact with a running container, like a terminal.
    a. Start an interactive bash(shell) session with an ubuntu:18.04 as a base image. Explore around the container using the shell. **What is different and what is same as an ubuntu VM?**
        i. Docker 'run' command to start an interactive BaSH session
    b. Get a shell into an already running container
5. Let us now create our own Docker image using Ubuntu:18.04 as a base image. The image you will create will run a simple C program, after installing the GCC compiler. The Dockerfile must:
    a. Specify the base image as **Ubuntu:18.04**
    b. Copy the program.c file from your instance to the docker image.
    c. Update the "apt" repository and install the GCC compiler
    d. Compiler the C program.
    e. Run the ./a.out *command*

    Use the following C program as a base, only modify your SRN:

```c
#include<stdio.h>

int main() {

    printf("Running this inside a container !\n");

    printf("My SRN is <YOUR SRN HERE>\n");

}
```

6. After you have your Dockerfile, build the image using docker build and run the container.

References to help you get started:

- How do you write a Dockerfile?
- Docker Build: A Beginner's Guide to Building Docker Images
- How to Create a Docker Image From a Container

Pull the following images form docker:
docker pull ubuntu:18.04
docker pull nginx:latest
docker pull python:latest
docker pull mongo:latest

Inside a folder task2 rn the following steps:
mkdir task2
cd task2

Create a simple C program:
nano prog.c

Copy the following contents into prog.c:

```
#include <stdio.h>


int main()
{
	printf("\n\nRunning this inside a container!\n\n");
	printf("My SRN is PES1201800046\n\n");
}
```

Create the Dockerfile:
nano Dockerfile

Copy the following contents into Dockerfile:

```
# Using ubuntu 18.04 image
FROM ubuntu:18.04

# Copying the C program
COPY prog.c /

# Set cprog as working directory
WORKDIR /

# Update apt
RUN apt-get update

# Install gcc
RUN apt install build-essential -y
RUN apt install manpages-dev -y

# Compile prog.c
RUN gcc prog.c -o prog.o

# Execute prog.o
CMD ["./prog.o"]
```

Build the docker image and add the tag as task2:
docker build -t task2 .

Check if the docker image was built correctly:
docker images

Run the container:
docker run task2

Task 3: Exposing ports,docker networks

Containers are also used to run web applications , they can be web servers such as Apache or Nginx, or web applications using REST APIs or any other web application. Similar to *Security Groups* for EC2 instances, we need to *expose* the right ports to access the container's web apps.

1. Download the sample HTML file from the Lab 5 drive folder , and modify your SRN.
2. Create a Dockerfile and then a docker image having an **nginx:latest** base image, and copying the html file into the default folder in the container. Build the docker image and **tag/name it your SRN**.
3. Run the docker container using the previously created docker image. Expose the HTTP port and check connectivity.
    a. Publishing/Exporting ports in a docker container
    b. Docker Container Tutorial #8 Exposing container ports

Hint: You are running a container on an EC2 instance. After exposing the port on the container , you might be able to access the nginx web server from within the EC2 instance, but not from the internet(your system). What needs to be configured for the AWS EC2 instance running your container?

In lab 3 , we saw VPC networks and how we can create a virtual network connectivity between virtual machines(EC2 instances). Similar networks need to be created for containers. We will explore connectivity without docker networks and see how docker networks make it much easier to connect within containers. To demonstrate this we will create a simple application using a python client and a MongoDB NoSQL server.

Note: You don't need to know how to use mongodb, code to use mongodb has been provided to you.

1. Run the mongodb container in a detached mode, exposing the default port(27017) of mongodb.
    a. Use the **mongo:latest** image.
    b. Name the container as **mongodb** while running the container. Naming a container
    c. Docker detached mode

2. Download sample.py from the drive folder. Modify the SRN wherever mentioned.

3. The MongoDB container is running, but we need to find out the *IP address* of the mongodb container. Find this IP using the docker inspect command. Modify this IP address in the sample.py file.
    a. Get a containers IP address
4. Create a Dockerfile, which:
    a. Uses **python** base image
    b. Updates the apt-repository
    c. Installs **pymongo** using pip (pymongo 3.11.2).
    d. Copies the sample.py from the instance to the container.
    e. Runs the python *command* , to run the python file.
5. Build the docker image using the above Dockerfile and run the container.
6. You should see that the data was correctly inserted and fetched from the database container.

The above tasks showed the connectivity between 2 containers *without* a network. This can cause problems as every time a container is created it *could possibly* have a different IP address. This is the issue *docker networks* tries to solve.

Docker Networking Options

Docker Networking | Docker bridge network deep dive | Container bridge drive

1. Create a docker bridge network , called **my-bridge-network.**
2. Stop the mongodb container you had created before and delete(rm) it. Run a mongodb container again, but now with the following parameters;
    a. network : **my-bridge-network**
    b. name: **mongodb**
    c. Exposed ports: **27017**
    d. Image: **mongo:latest**
3. Go back to sample.py , comment line 3 and uncomment line 4. We are now going to use the name of the database containers as the host name, leaving the ip address resolution to docker.
4. Build the python app docker image again as you did previously and run the container using the built image. You should see that insertion and retrieval have been done successfully.

Expose the correct ports to access the container's web app pages that is running on the nginx server:

mkdir task3a
cd task3a

Create my.html:
nano my.html

Copy the html file contents:

```
<html>
	<body>
		<h1>My SRN is PES1201800046</h1>
		<h2>I am running a nginx container!</h2>
	</body>
</html>
```

Create the Dockerfile
nano Dockerfile

Copy the following contents into the Dockerfile:

```
# Using nginx image
FROM nginx:latest


# Copy the html file
COPY my.html /usr/share/nginx/html/my.html
```

Build the container
docker build -t pes1201800046 .

Check if the image is present
docker image

Run the container by exposing the correct ports for http i.e. 80 & 80
docker run -p 80:80 pes1201800046

Go to the webpage: http://ec2-44-192-114-93.compute-1.amazonaws.com/my.html to view the my.html file being displayed

Running a simple application using python & mongodb sever-client using IP address of the running container: **cd, mkdir task3, cd task3**

Run the mongodb server in the detached mode by exposing the correct ports for mongodb - 27017

docker run -dp 27017:27017 mongo:latest

Find the container ID of the currently running mongodb server:

docker ps

Find the correct IP address of the running container using its container ID:

docker inspect <container ID>

Create a sample.py file

nano sample.py

Add the following contents into sample.py, add the IP address as the host
from pymongo import MongoClient

```
from pymongo import MongoClient


host = MongoClient("<IP HERE>")
#host = MongoClient("mongodb")


db = host["sample_db"]
collection = db["sample_collection"]


sample_data = {"Name:":"Ruchika","SRN":"PES1201800046"}
collection.insert_one(sample_data)
print('Inserted into the MongoDB database!')


rec_data = collection.find_one({"SRN":"PES1201800046"})
print("Fecthed from MongoDB: ",rec_data)
```

Create the Dockerfile:
nano Dockerfile

Add the following contents into the Dockerfile:

```
# Using python image
FROM python

# Update packages
RUN apt-get update -y

# Install mongodb using pip
RUN pip install pymongo

# Copy the python file
COPY sample.py sample.py

# Execute the python file
CMD ["python", "sample.py"]
```

Build the dockerfile:
docker build -t task3 .

Run the container:
docker run task3
(We can see that python application successfully wrote and read from the MongoDB database)
Running a simple application using python & mongodb sever-client using a newly created docker network:

Find the container ID of the currently running mongodb server:
docker ps

Stop the currently running mongodb server container:
docker stop <container ID>

Create a new docker network:
docker network create my-bridge-network

Change the host in sample.py to mongdb in the line - host = MongoClient("mongodb")
nano sample.py

Rebuild the python app image:
docker build -t task3 .

Run the mongodb server on the newly created docker network and exposing the correct port for mongodb - 27017 and tag the container to the name - mongodb on a detached port
docker run -dp 27017:27017 --name mongodb --network=my-bridge-network mongo:latest

Check if the mongodb server is running in the background:
docker ps
Run the python app in the same docker network:
docker run --network=my-bridge-network task3
(We can see that python application successfully wrote and read from the MongoDB database)

Task 4: Docker compose

Till now, we have been using docker commands to start and stop containers,create docker networks etc. But this becomes challenging in a real environment when a single application may have 100s of containers(microservices) that need to talk to each other within a single network. While docker networks solve this problem partially, it still doesn't solve the issue of having to run multi-container applications and to automatically bootstrap these containers into the same application, this is where *Docker Compose* comes into the picture.

Docker compose helps bootstrap complex multi container applications, helps maintain the dependencies between them , create a network among these containers and even help scale the entire application entirely or particular containers within the application.

Overview of Docker Compose

"What, Why, How" Docker Compose?

We will use the same python-mongodb application to use docker compose and even scale containers within the application automatically. Docker compose uses YAML Files to specify the different *pieces(containers)* of the application, along with the required configurations. YAML files are EXTREMELY popular when it comes to bootstrapping complex applications not only in Docker!

Docker Compose in 12 Minutes

1. For the purposes of this lab make sure all the following files are in the same directory
    a. docker-compose.yml
    b. Dockerfile
    c. sample.py
2. Install docker compose using the following link:
    a. Install compose on linux systems
3. Go to the docker-compose.yml file and try to understand the syntax and what each line does.
4. Within the same directory, run: **docker-compose up**.
    a. Docker Compose command-line reference

    What you see is that Docker compose has built your python application, started the mongodb server , created links internally between the containers (network) and started both the containers together as a *unified application*. The python container exits, since its done with its utility of writing and reading to the database.

5. Now we will scale the python application, so that we have 3 containers of the python application, but keep only one container of the mongodb.
    a. docker-compose scale - Use this link as a reference and scale **only the python application. Use the correct "service" to scale**

Docker compose:

```
cd
mkdir task4
cd task4
```

Install docker-compose package dependencies:

```
sudo apt update -y
sudo apt install python3-pip -y
sudo apt install python3-dev -y
sudo apt install libffi-dev -y
sudo apt-get install libssl-dev -y
sudo apt install gcc -y
sudo apt install libc-dev -y
curl https://sh.rustup.rs -sSf | sh
source $HOME/.cargo/env
sudo apt install cargo -y
sudo apt install make -y
```

Install docker-compose from its official releases:
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

Check if docker-compose and its correct version has been installed:
docker-compose --version

Create a sample.py file
nano sample.py

Add the following contents into sample.py, add the IP address as the host

```
from pymongo import MongoClient

host = MongoClient("mongodb")


db = host["sample_db"]

collection = db["sample_collection"]


sample_data = {"Name:":"Ruchika","SRN":"PES1201800046"}

collection.insert_one(sample_data)

print('Inserted into the MongoDB database!')


rec_data = collection.find_one({"SRN":"PES1201800046"})

print("Fecthed from MongoDB: ",rec_data)
```

Create the Dockerfile:
nano Dockerfile

Add the following contents into the Dockerfile:

```
# Using python image

FROM python

# Update packages

RUN apt-get update -y

# Install mongodb using pip

RUN pip install pymongo

# Copy the python file

COPY sample.py sample.py

# Execute the python file

CMD ["python", "sample.py"]
```

Create docker-compose.yml
nano docker-compose.yml

Add the following contents into docker-compose.yml

```
services:

  pycode:

    build: .

    links:

      - mongodb

  mongodb:

    image: mongo

    ports:

      - 27017
```

Build the docker compose application:
docker-compse build

Run the python-mongodb application as a docker compose application:
docker-compose up

Scale only the python applications by using docker-compose:
docker-compose up --scale pycode=3

Task 5: Docker volumes

Docker volumes are an important and crucial concept. Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. Volumes are either created dynamically by Docker or you can mount the host directory. In this task,we will focus on mounting a host volume.

Docker bind mount | Sharing data between host and Container

1. Create a new directory on the EC2 host , with the name as your SRN.
2. Add the following folders inside it:
    a. sample.py (A simple python application which reads from a text file called details.txt and prints them line by line)
    b. A text file called details.txt containing your name,srn,section,semester.
3. We will be using **Ubuntu:18.04** as a base image.
4. For the ubuntu container, mount the host volume you have created and create a bash shell into the container. Start a container with a bind mount
5. Inside the container run the python program to display contents of the list.

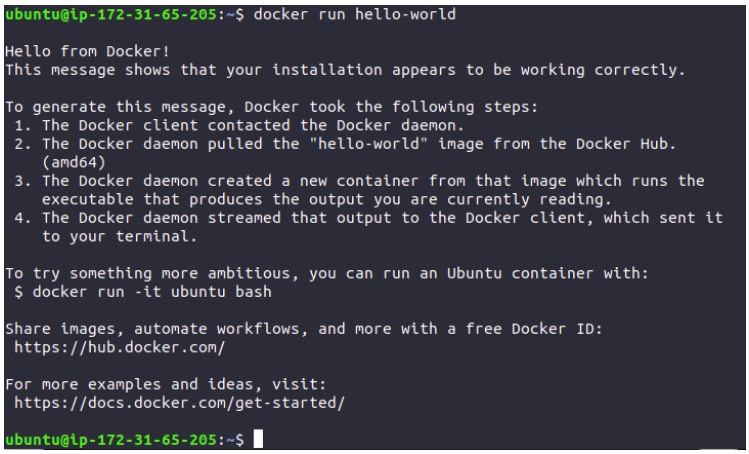
Additional Resources/Common bugs you might encounter:

1. What is Docker?
2. docker command not found even though installed with apt-get
3. docker CLI & Dockerfile Cheat Sheet
4. docker cheat sheet
5. Not able to reach the container(for web apps)? Think about EC2 security groups.
6. 9 Common Dockerfile Mistakes - Runnablog
7. How to debug and fix common docker issues.
8. Not able to build/run docker containers due to insufficient space on EC2: How To Remove Docker Images, Containers, and Volumes
9. Docker - Container is not running
10. exited with code 0 docker
11. Docker container has a name conflict? Remove all stopped containers
12. Curious about MongoDB and what it is ?
    a. What is NoSQL? NoSQL Databases Explained
    b. What Is MongoDB?
    c. Tutorial — PyMongo 3.11.2 documentation

LAB 5 SCREENSHOTS

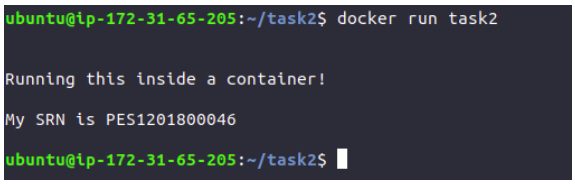
Task 1: Installing Docker Engine on AWS EC2 instance

1a.png - Screenshot of running docker hello-world



Task 2: Docker images and Dockerfiles

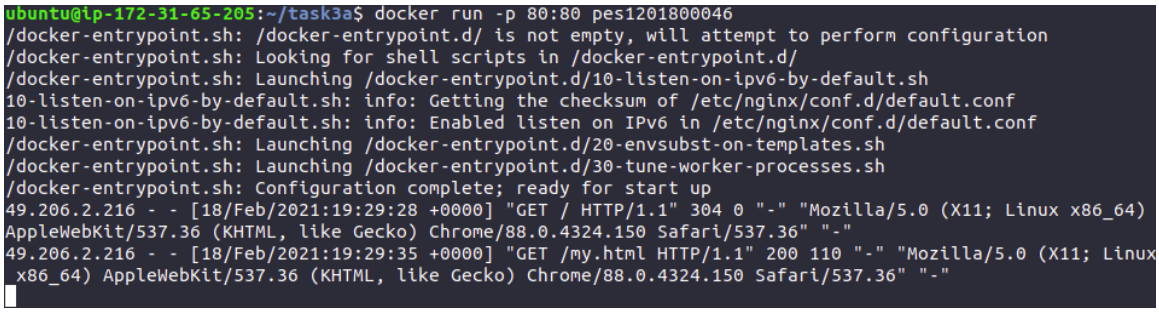
2a.png - Screenshot of C Program successfully run inside container



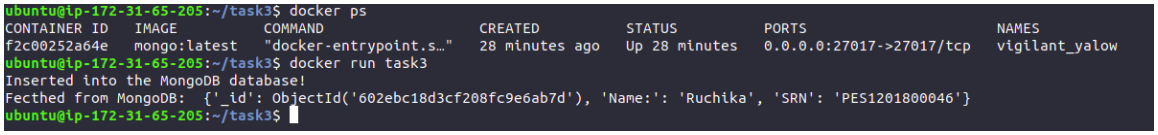
Task 3: Exposing ports and docker networks

3a.png - Screenshot of sample.html showing the web page on the browser and public dns of the EC2 host of docker

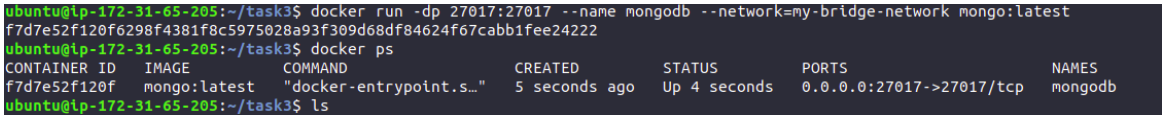
3b.png - Screenshot of docker container running nginx on the terminal of EC2 host



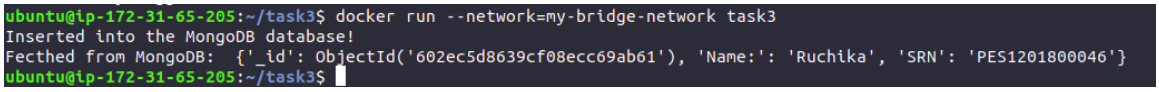
3c.png - Screenshot of python application successfully writing and reading from the MongoDB database



3d.png - Screenshot showing mongodb being run within network using the docker command



3e.png - Screenshot showing python file being run within network and successfully writing and reading from MongoDB using the docker command



Task 4: Docker compose

4a.png - Screenshot of python-mongodb application running as a docker compose application from the logs of the application by running the commands:

docker-compse build
docker-compose up

4b.png - Screenshot of 3 python application writes and reads from MongoDB after scaling the python application by running the command:

docker-compose up --scale pycode=3

Lab 6 - RabbitMQ and Docker-Compose

RabbitMQ

RabbitMQ is an open-source message-broker software that originally implemented the Advanced Message Queuing Protocol and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol, MQ Telemetry Transport, and other protocols.

What can RabbitMQ do for you?

Messaging enables software applications to connect and scale. Applications can connect to each other, as components of a larger application, or to user devices and data. Messaging is asynchronous, decoupling applications by separating sending and receiving data.

You may be thinking of data delivery, non-blocking operations or push notifications. Or you want to use publish / subscribe, asynchronous processing, or work queues. All these are patterns, and they form part of messaging.

RabbitMQ is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

Docker-Compose

Compose is a tool for defining and running multi-container **Docker** applications. With **Compose**, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Objective

In this lab experiment we will observe how two of the important messaging technique from RabbitMq work, Namely:

1. Work Queues
2. Publish/Subscribe

We will run the messaging application's and the RabbitMq server as the microservices using docker-compose. Source files can be found at this link.

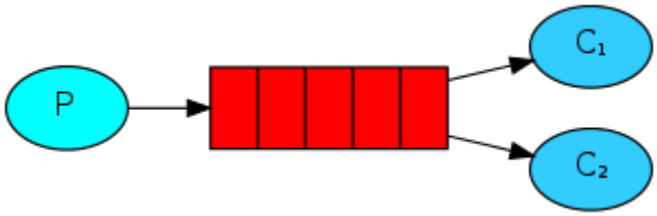
Prerequisites:

- Docker Installed on your system. (Refer to Task1 steps in the Lab5 manual)
- Understand how to write yaml files.

TASK-1 (Work Queues)

In this task we will implement the "Work Queues" Message brokering service. We are going to have three services running using Docker compose.

The main idea behind Work Queues (aka: Task Queues) is to avoid doing a resource-intensive task immediately and having to wait for it to complete. Instead we schedule the task to be done later. We encapsulate a task as a message and send it to the queue. A worker process running in the background will pop the tasks and eventually execute the job. When you run many workers the tasks will be shared between them.

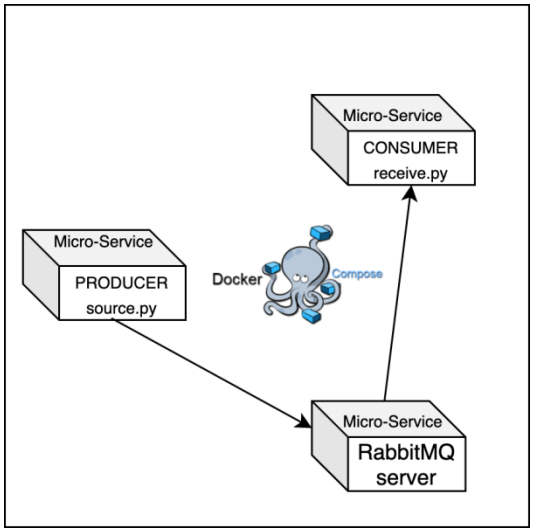


Background:

We will build a multi-container application having **three services**. One service will run your RabbitMq Server, Other two will be your Producer service and worker service(consumer).

The Architecture:

1. The producer will Run a Python Application which will push the message to the Queue.
2. The producer will push a certain amount of messages on the Queue.
3. The consumer will run a Python Application which will pop a message from the Queue and print it.
4. Scaling Multiple workers (consumers), the messages will be shared among the workers(consumers).



You are already provided with the application file for the producer service. A producer service run's an application "source.py". The producer service will push the message into the queue by connecting to the RabbitMq service running in the same docker-compose network.

What will you be doing:

- Build the python application for the worker service (consumer service)
- Write the YAML file to build the services.
- Scale the Number of workers while you do docker-compose.

sudo docker-compose build
sudo docker-compose up --scale consumer=4

```
# YAML file - PES1201800046: Ruchika Shashidhara
version: '3.1'

services:
  rmq:
    image: rabbitmq:3.8.3-alpine

  consumer:
    build:
      context: receive
      dockerfile: Dockerfile

    command: sh -c "sleep 15 && python receive.py"

    links:
      - rmq

    depends_on:
      - rmq
    restart: on-failure

  producer:
    build:
     context: send
      dockerfile: Dockerfile

    command: sh -c "sleep 20 && python source.py"

    links:
      - rmq

    depends_on:
      - rmq
    restart: on-failure
```

Note:

- please download the source folder given to you
- The source folder will have two directories namely “send” and “receive’. It will also consist of one empty yml file. Please maintain the directory structure given to you.
- The “send” and “receive” folder will have a Dockerfile.
- The “send” folder has “source.py” file.
- The “receive” folder does not has the “receive.py” file.
- We are gonna use the terms “consumer” and “workers” Interchangeably.

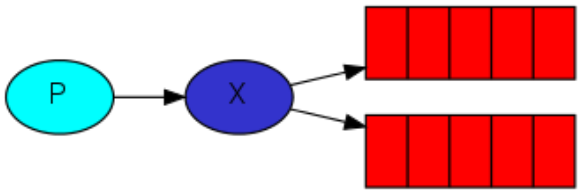
Steps:

- Build the python application for workers(consumer service).
- The worker (consumer) application should pop the message available in the queue and print it.
- The idea is when we will have multiple workers running(consumers) the messages will be shared among the workers.
- You can get the code for the worker application from here. Make sure you name the file as ‘receive.py” and change the queue name according to what’s given in “source.py” file. Also change the hostname by looking into “source.py”.
- Add the “receive.py” file to receive folder.
- Edit your yml file and add the below mentioned requirements. yaml file will have the skeleton code for you to begin with.
- Source files are present at this link, download & make the above changes.
- You need to have three services one service is your rabbitmq server, other two are your producer and consumer. Below you can find the service specifications.
    - RabbitMQ service
        - Service Name: rmq
        - image: rabbitmq:3.8.3-alpine
    - Producer service
        - Service Name: producer
        - build context: “send”
        - Dockerfile: Use the dockerfile inside the send folder
        - command: sh -c "sleep 20 && python source.py" (would suggest to use the exact same command)
        - links: rmq
        - dependency: rmq
    - Consumer service
        - Service Name: consumer
        - build context: “receive”

    - Dockerfile: Use the dockerfile inside the receive folder
    - command: sh -c "sleep 15 && python receive.py" (would suggest to use the exact same command)
    - links: rmq
    - dependency: rmq

- The producer service should use the Dockerfile present inside the send folder.
- The consumer service should use the Dockerfile present inside the receive folder.
- Understand the concept of context.
- Also edit the message being sent as the Message should Include your **SRN** in “source.py”.
- Now run your docker services by scaling the number of consumers to **4.**
- Identify what command to run in order to scale the services while you start them.
- We are scaling the consumer service to create 4 workers. You can observe that the message sent by the sender is shared by the workers(consumers). Take the required SS.

TASK-2 (Publish/Subscribe)



The core idea in the messaging model in RabbitMQ is that the producer never sends any messages directly to a queue. Actually, quite often the producer doesn't even know if a message will be delivered to any queue at all.

Instead, the producer can only send messages to an exchange. An exchange is a very simple thing. On one side it receives messages from producers and the other side it pushes them to queues. The exchange must know exactly what to do with a message it receives. Should it be appended to a particular queue? Should it be appended to many queues? Or should it get discarded. The rules for that are defined by the exchange type.

Background:

We will build a multi-container application having **three services** just like the previous task. One service will run your RabbitMq Server, Other two will be your Producer service and worker service(consumer).

The Architecture:

1. The producer will publish messages to the exchange on the RabbitMq server.
2. The producer will Run a Python Application which will publish the message to the exchange.
3. The consumer will run a Python Application which is subscribe to the exchange using Queues.
4. Scaling to Multiple workers (consumers), All the messages will be received by all the consumers.

sudo docker-compose build
sudo docker-compose up --scale consumer=5

Steps:

- Please find the “emit_log.py” and ”receive_log.py” from this link.
- Understand what each of the applications does.
- Your producer service should run the “emit_log.py” Application.
- Your consumer service should run the ”receive_log.py” Application.
- You can use the previous yaml file for this task.
- Just replace the old application files with the new application files.
- Modify the yaml file to run the new application’s.
- Modify the application’s to connect to the RabbitMq server. The producer application should push your “SRN'' as the message.
- You need to start the services and make sure you scale the consumer service while you start & you need to scale the consumer service to **5**. Take appropriate SS.
- If you are using the same “Source” directory for this task, then make sure you delete your Images and Containers because if you run the service after modifying the application files it won’t build the new image for your services. Or you can refer to this process.

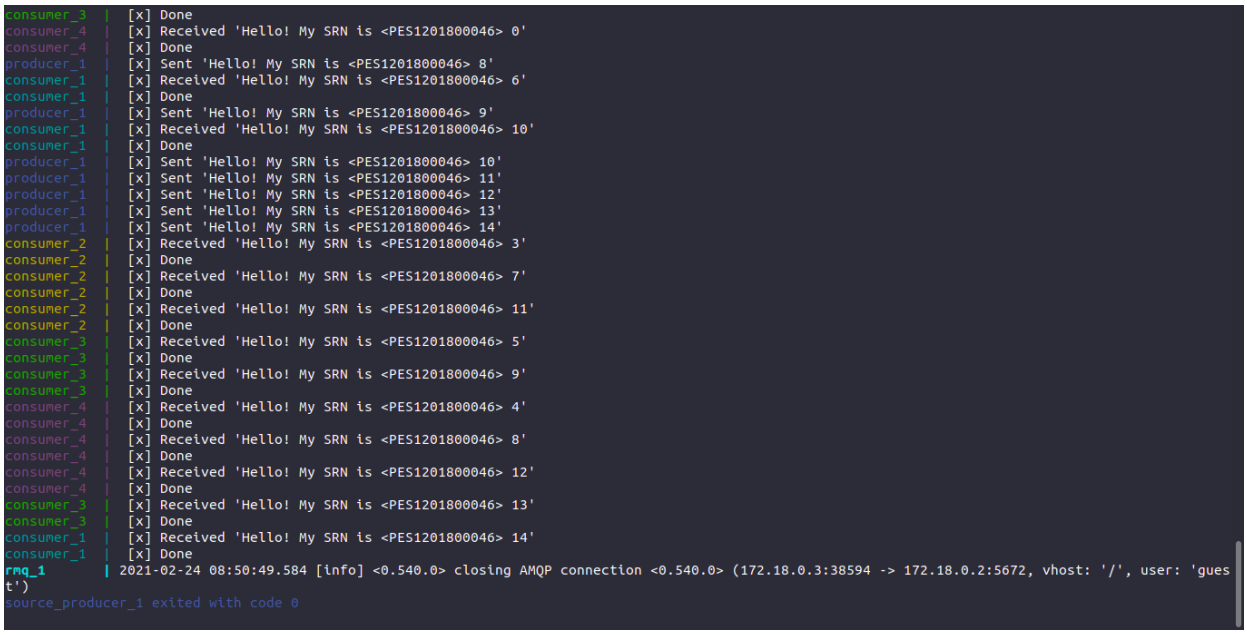
Reference:

- Hint: We will be using the same application files provided in this link: https://www.rabbitmq.com/tutorials/tutorial-two-python.html
- Docker-compose and scaling: https://medium.com/@karthi.net/how-to-scale-services-using-docker-compose-31d7b83a6648

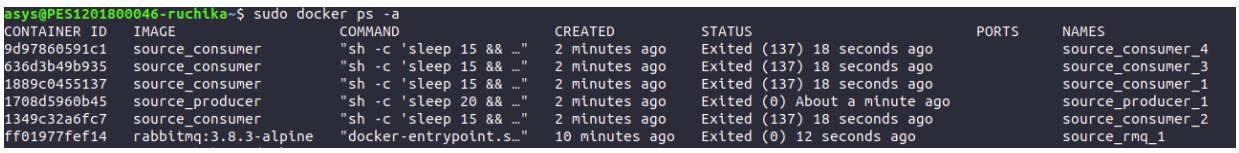
LAB 6 SCREENSHOTS

Task 1: Work Queues

1a.png - Docker Compose Output



1b.png - Docker ps output showing which containers are running



Task 2: Publish - Subscribe

2a.png - Docker Compose Output

```
rmq_1       | 2021-02-24 09:07:05.445 [info] <0.331.0> Running boot step direct_client defined by app rabbit
rmq_1       | 2021-02-24 09:07:05.445 [info] <0.331.0> Running boot step os_signal_handler defined by app rabbit
rmq_1       | 2021-02-24 09:07:05.445 [info] <0.451.0> Swapping OS signal event handler (erl_signal_server) for our own
rmq_1       | 2021-02-24 09:07:05.725 [info] <0.9.0> Server startup complete; 0 plugins started.
rmq_1       |  completed with 0 plugins.
rmq_1       | 2021-02-24 09:07:08.145 [info] <0.457.0> accepting AMQP connection <0.457.0> (172.18.0.3:39978 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:08.155 [info] <0.457.0> connection <0.457.0> (172.18.0.3:39978 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
consumer_1  |  [*] Waiting for logs. To exit press CTRL+C
rmq_1       | 2021-02-24 09:07:08.190 [info] <0.475.0> accepting AMQP connection <0.475.0> (172.18.0.6:42672 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:08.195 [info] <0.475.0> connection <0.475.0> (172.18.0.6:42672 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
consumer_4  |  [*] Waiting for logs. To exit press CTRL+C
rmq_1       | 2021-02-24 09:07:08.262 [info] <0.491.0> accepting AMQP connection <0.491.0> (172.18.0.5:43828 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:08.266 [info] <0.491.0> connection <0.491.0> (172.18.0.5:43828 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
consumer_2  |  [*] Waiting for logs. To exit press CTRL+C
rmq_1       | 2021-02-24 09:07:09.085 [info] <0.508.0> accepting AMQP connection <0.508.0> (172.18.0.4:57900 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:09.089 [info] <0.508.0> connection <0.508.0> (172.18.0.4:57900 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
consumer_5  |  [*] Waiting for logs. To exit press CTRL+C
rmq_1       | 2021-02-24 09:07:09.189 [info] <0.524.0> accepting AMQP connection <0.524.0> (172.18.0.7:38750 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:09.193 [info] <0.524.0> connection <0.524.0> (172.18.0.7:38750 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
consumer_3  |  [*] Waiting for logs. To exit press CTRL+C
rmq_1       | 2021-02-24 09:07:14.055 [info] <0.545.0> accepting AMQP connection <0.545.0> (172.18.0.8:60846 -> 172.18.0.2:5672)
rmq_1       | 2021-02-24 09:07:14.058 [info] <0.545.0> connection <0.545.0> (172.18.0.8:60846 -> 172.18.0.2:5672): user 'guest' authenticated and gran
ted access to vhost '/'
producer_1  |  [x] Sent ' Hello, my SRN is PES1201800046! '
consumer_4  |  [x] b' Hello, my SRN is PES1201800046! '
consumer_1  |  [x] b' Hello, my SRN is PES1201800046! '
consumer_2  |  [x] b' Hello, my SRN is PES1201800046! '
consumer_5  |  [x] b' Hello, my SRN is PES1201800046! '
consumer_3  |  [x] b' Hello, my SRN is PES1201800046! '
rmq_1       | 2021-02-24 09:07:14.068 [info] <0.545.0> closing AMQP connection <0.545.0> (172.18.0.8:60846 -> 172.18.0.2:5672, vhost: '/', user: 'gues
t')
source_producer_1 exited with code 0
```

2b.png - Docker ps output showing which containers are running

```
asys@PES1201800046-ruchika~$ sudo docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED             STATUS                          PORTS     NAMES
483646b0f587   source_consumer        "sh -c 'sleep 15 && …"   About a minute ago   Exited (137) 14 seconds ago               source_consumer_3
ec3c0cf6404c   source_consumer        "sh -c 'sleep 15 && …"   About a minute ago   Exited (137) 14 seconds ago               source_consumer_4
d7cbb212e605   source_consumer        "sh -c 'sleep 15 && …"   About a minute ago   Exited (137) 14 seconds ago               source_consumer_2
bc5d0ccf763d   source_consumer        "sh -c 'sleep 15 && …"   About a minute ago   Exited (137) 14 seconds ago               source_consumer_1
8bbfc74480d0   source_consumer        "sh -c 'sleep 15 && …"   About a minute ago   Exited (137) 14 seconds ago               source_consumer_5
a2d18a097884   source_producer        "sh -c 'sleep 20 && …"   About a minute ago   Exited (0) About a minute ago             source_producer_1
ff01977fef14   rabbitmq:3.8.3-alpine  "docker-entrypoint.s…"   26 minutes ago       Exited (0) 9 seconds ago                  source_rmq_1
```

Lab 7 - Jenkins (DevOps, CI/CDtool)

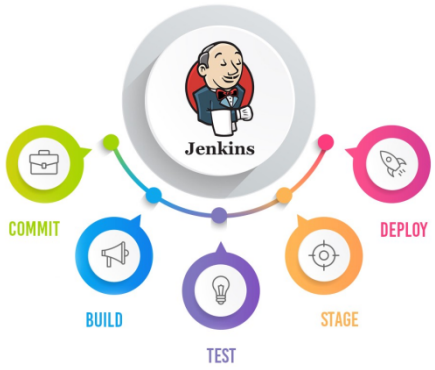
Introduction:

Welcome to Lab 7! This simple exercise is designed to introduce you to Jenkins and continuous integration.

What is Jenkins?

Jenkins is an extensible, open source continuous integration server. It builds and tests your software continuously and monitors the execution and status of remote jobs, making it easier for team members and users to regularly obtain the latest stable code.

What is Jenkins Pipeline?



In simple words, Jenkins Pipeline is a combination of plugins that support the integration and implementation of continuous delivery pipelines using Jenkins.The pipeline as Code describes a set of features that allow Jenkins users to define pipelined job processes with code, stored and versioned in a source repository.

Overview of the Experiment

- Setup jenkins Using Docker.
- Set up a job in Jenkins to connect to your repository and build C++ hello.cpp.
- Set up second job to run the program after the build completes.

Prerequisite:

- Docker Installed on your system. (Refer to Task1 steps in the Lab5 manual)

Task-1

Aim: Set up Jenkins using Docker.

Steps:

Dockerfile

```
FROM jenkins/jenkins:lts
# if we want to install via apt
USER root
RUN apt-get update && apt-get install -y make && apt-get install -y g++
# drop back to the regular jenkins user - good practice
USER jenkins
```

- Put the above contents into a Dockerfile in a folder and open a terminal in that folder.
- Build the dockerfile using this command: `"sudo docker build -t jenkins:lab7 . "`
- Run your container using this command `"sudo docker run -p 8080:8080 -p 50000:50000 -it jenkins:lab7" (Note down the password shown on the terminal)`
- Open URL: localhost:8080 on your browser.
- Enter the password shown on your terminal after running the container (You can set the password to ADMIN later).
    - In case you did not note down the password displayed on the terminal, you can find the password by connecting to the container (via "sudo docker exec –it <container_id> /bin/bash") and checking the file /var/Jenkins_home /secrets/initialAdminPassword inside the container
- **Integrate GitHub to Jenkins**: When prompted for plugin installation, click on "Select Plugins to Install" and then search for Github and check the github option. (This step may take a few minutes to complete)
- Take the necessary SS.

Task-2

Aim: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp.

Steps:

- Navigate to Jenkins server. Click ***New Item***.
- Enter a name for your project, click *Freestyle Project*, then *OK*. *Note*: Please do not include a space.
- Name this something unique so there are no collisions.
- Select github project and Enter the repository URL. Use this repository link: https://github.com/sujeeth-cc/Lab7-Jenkins.git
- Set up *Source Code Management*, Select *git*. Enter the URL of git repository.
- Add "`*/main`" in **Branches to build** (Do not delete the existing */master branch)
- Setting up *Build Triggers.* Select *Poll SCM*.
- Set up cron job by putting in "`H/2 * * * *`" in the Schedule box
- Set up *Build*. In **Add build step** pull-down menu, select *Execute Shell*.
- Enter "`make -C original`"(This will run the Makefile).
- Click *Save*.
- Click on build now.
- Take the required SS.

Task-3

Aim: The final step to this exercise is to set up a second job that automatically runs after the project builds. This is different from the other job because this will not have a git repository - it doesn't even build anything.

Just a note: In a real-life scenario you wouldn't run a program through a build job just like this because I/O is not possible via this console. There are other tools people use at this step like SeleniumHQ, SonarQube, or a Deployment. The point of this is to show downstream/upstream job relationships.

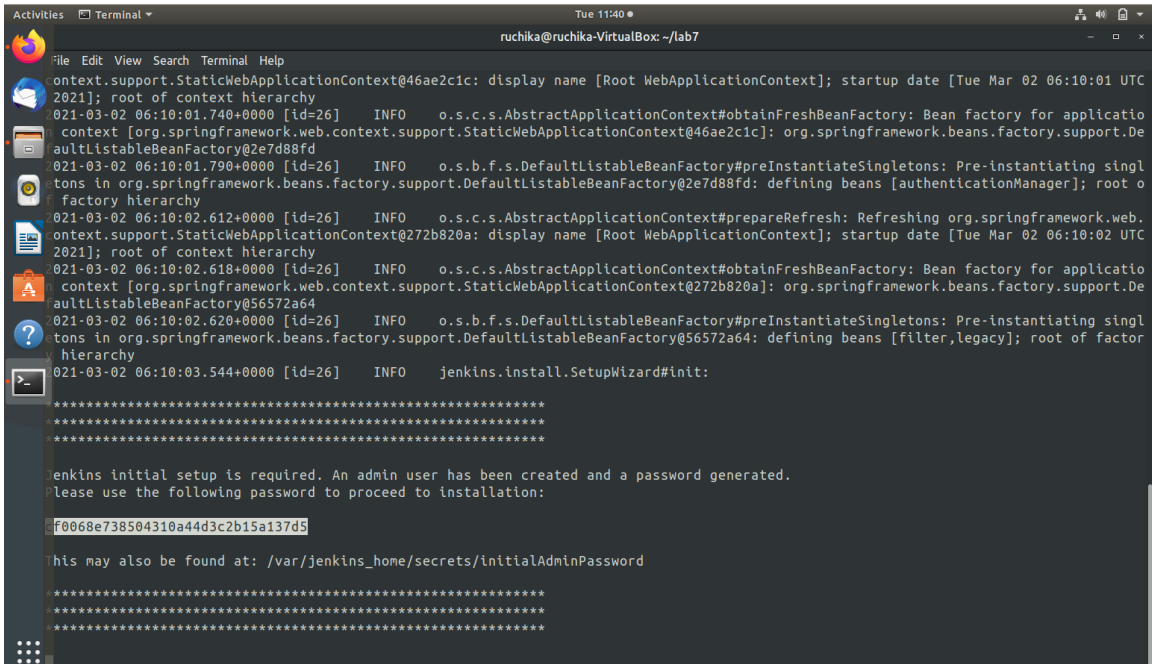
Steps:

- Create a new Job in Jenkins, Click *New Item* in the left panel
- Enter a name for your second job, click *Freestyle Project*, then *OK*.
- Go immediately to build step and select *Execute Shell*.
- Enter the following Command `/var/jenkins_home/workspace/<the name of your first project>/original/hello_exec`
- Save
- Set your first job to call the second.
- Go to your first job (i.e item) and open the *Configure* page in the pull down menu
- Scroll to bottom and add a Post-Build Action. Select ***Build other projects***.
- Enter the name of your second job.
- Save.
- Run your first job.
- Do this by clicking build now on the main page.
- After that successfully builds, go and check your second job. You should see it successfully run.
- Select a Build Job from History and go to the console log to see your program output. If you program has run there then you successfully set up a basic pipeline.
- Take the required SS.

LAB 7 SCREENSHOTS

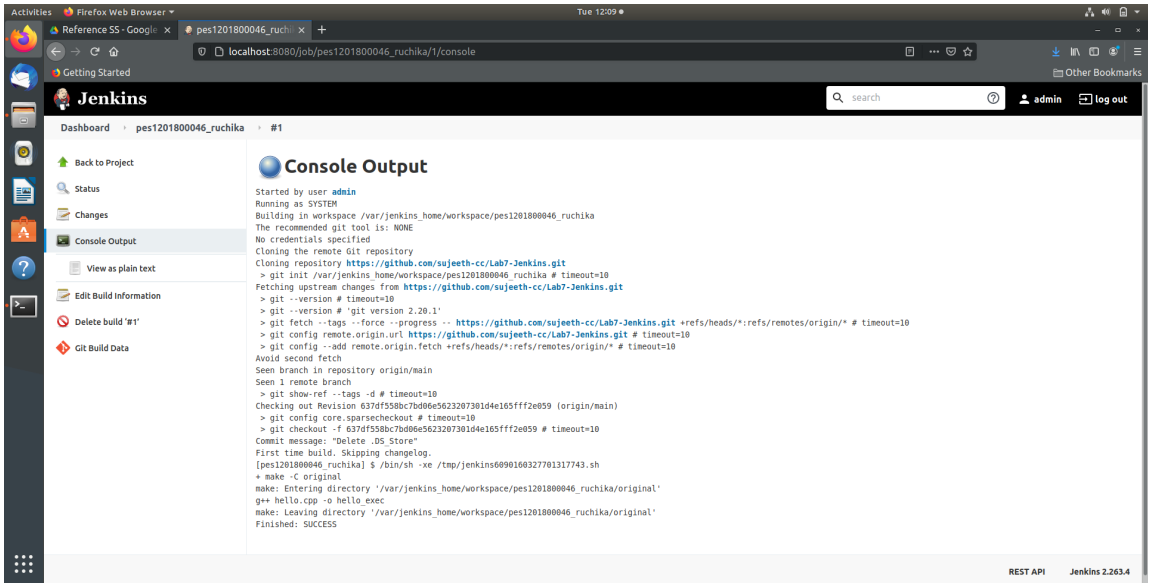
Task 1: Set up Jenkins using Docker

1A.png - SS of the running Docker Container

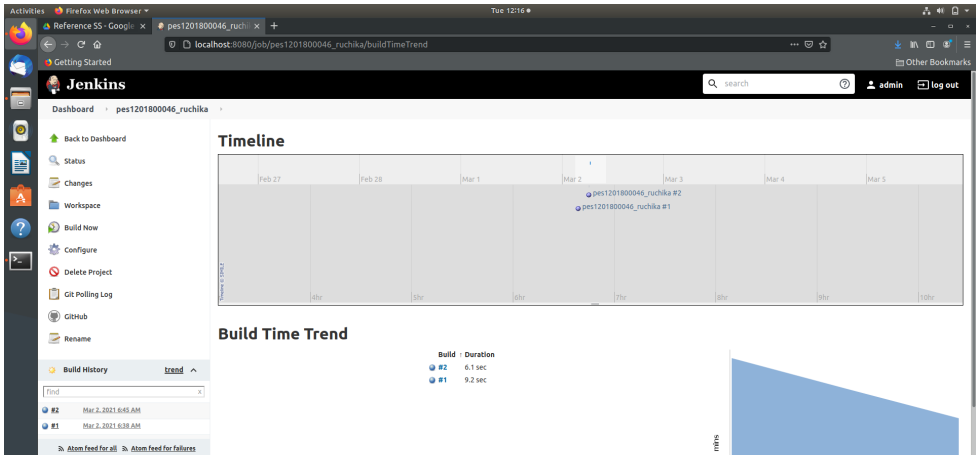


Task 2: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp

2A.png - SS of the console output showing build is successful of first job

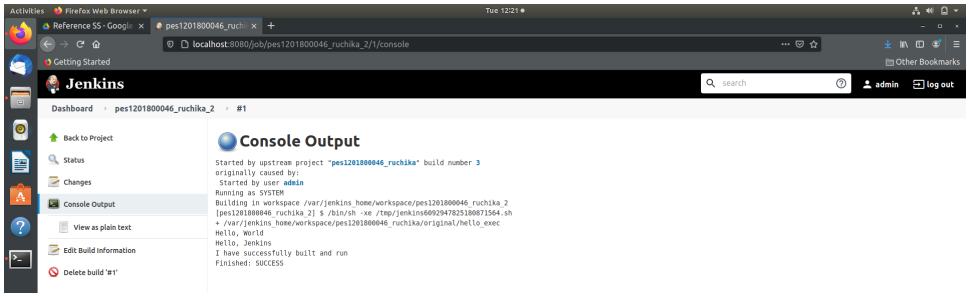


2B.png - SS showing Build History under Build Time Trend

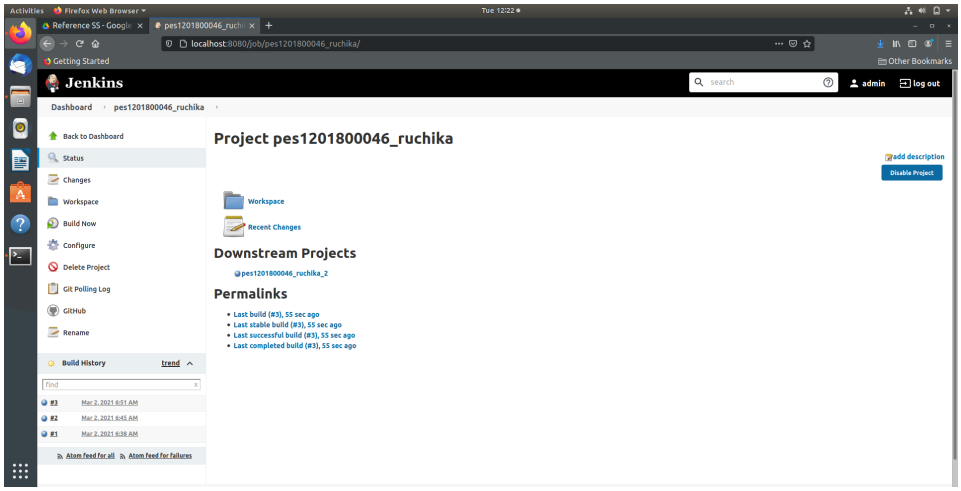


Task 3: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp

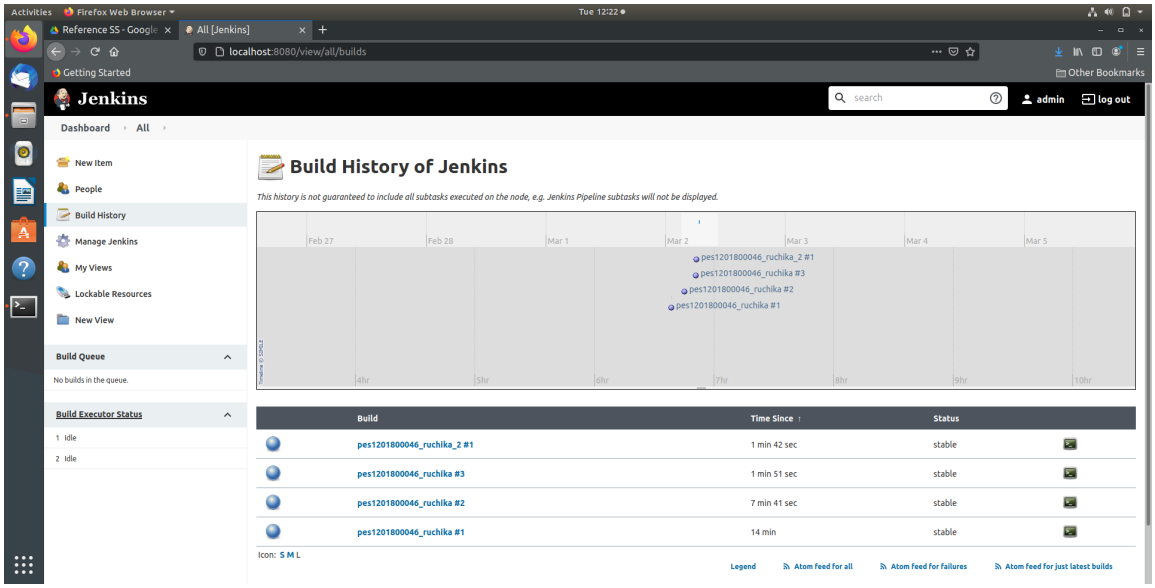
3A.png - SS of the console output showing build is successful of the second job



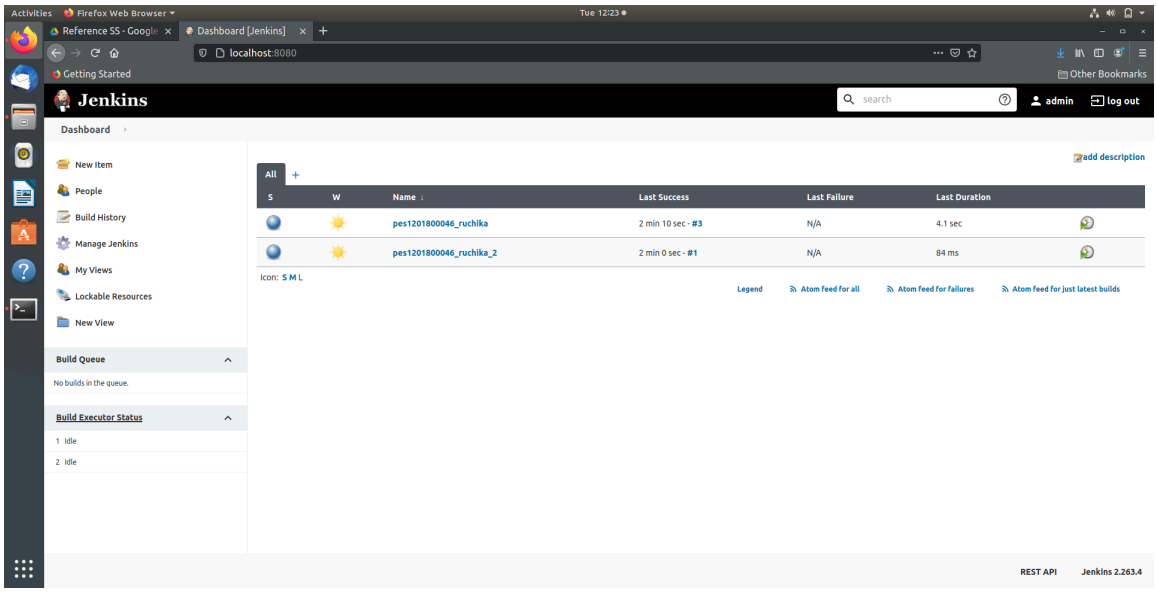
3B.png - SS of the status page of the first job



3C.png - SS of build history of Jenkins showing status table of the first & second jobs



3D.png - SS of Jenkins Dashboard showing the first & second jobs



Lab 8 - Kubernetes

Introduction to Kubernetes

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage:

- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again
- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies
- Automatically leveraging additional resources made available when a new host is added to the cluster
- Easily performing canary deployments and rollbacks

Important Kubernetes Terminologies:

1. Kubernetes Namespace - Kubernetes Documentation: Namespaces
2. Pod - Pods – Kubernetes Documentation, What is a Pod.
3. Replica Set - ReplicaSet
4. Deployment - Deployments
5. Service - Kubernetes Service
6. LoadBalancer Service - Kubernetes LoadBalancer Service
7. NodePort Service - Kubernetes NodePort Service
8. Ingress Controller - Ingress
9. Horizontal Pod Autoscaler - Kubernetes Horizontal Pod Autoscaler

Quick Points:

1. Ensure Docker is installed, up and running before using minikube.
2. Pods may take some time (a few minutes in a bad network) initially to start up, this is normal.
3. All resources mentioned in all the tasks must be created only in the **default Kubernetes namespace**, modifying other namespaces such as kube-system may cause Kubernetes to stop working.

4. If you are unable to show the results to your respective lab faculty, you can submit the screenshots to Edmodo but only in PDF or WORD format (and not in ZIP format)

TASK-1 (Create a Kubernetes cluster using minikube)

The first task is to set up a Kubernetes cluster and a command-line tool called "kubectl" to interact with this cluster. For this lab, we will be using *minikube* which is a VM/Docker-based tool that helps to create a Kubernetes cluster quickly to either test applications/learn Kubernetes.

Kubernetes being a container-orchestration service needs an engine to create/destroy containers and uses Docker for this purpose. A real Kubernetes cluster runs pods natively on the host machine using the docker engine. So all container processes are run natively on the host machine. Some examples of popular Kubernetes clusters are AWS Elastic Kubernetes Service(EKS), Google Kubernetes Engine(GKE), and kubeadm which is used to create clusters manually.

Minikube is slightly different from the real Kubernetes cluster, it creates a virtual machine/docker container and runs the pods inside the VM/container. This is done so that we can get a quick Kubernetes cluster up and running without bothering about the details of setting up one or to avoid cloud services. Details about how this impacts deployments/services will be clearly explained as we go on in the lab.

To set up minikube Kubernetes cluster, follow the following steps:

1. Install Docker/Ensure docker is already installed on the host machine. Make sure docker is up and running
2. Install Kubectl, the CLI tool used to interact with the Kubernetes cluster by following **Steps 1 to 4** under "**Install kubectl binary with curl on Linux**" in the following link:
   https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/ **or**
   https://kubernetes.io/docs/tasks/tools/install-kubectl/

Download the latest release
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

Download the kubectl checksum file
curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

Validate the kubectl binary against the checksum file, output should be kubectl: OK
echo "$(<kubectl.sha256) kubectl" | sha256sum --check

Install kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

Test to ensure the version you installed is up-to-date:
kubectl version --client

3. Download and install minikube by running the following 2 commands (each is a complete line of command:

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Ref Link - https://minikube.sigs.k8s.io/docs/start/#binary-download

4. Startup the minikube cluster by running: `minikube start`
    a. This might take a while initially as it needs to pull the docker image and set up kubectl
5. Test your kubectl is configured and correctly running, by running: `kubectl get node`

You should see a similar output:

```
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   65m   v1.18.3
```

TASK-2 (Create a Kubernetes pod)

All resources such as pods, deployments, services etc are created using YAML files. For this task and the following tasks, we will see how we can use Kubernetes to orchestrate Nginx web servers.

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. Open the **pod.yaml** file and examine the contents of the YAML and what each property/field does.

Contents of pod.yaml

```
apiVersion: v1
#Specifies the kind of resource we are creating
kind: Pod
#Metadata to identify the pod in the cluster
metadata:
  name: nginx
  labels:
       name: nginx
#The specifications of the container(s) within the pod
#A pod is an application, not a container, therfore a pod can container many containers
#Typically there is a main container and "side-car" containers
spec:
  containers:
  # The name of the container you are creating
  - name: nginx
  #The Docker image of the container
       image: nginx:latest
       resources:
       #Resource limits for the container
       limits:
       memory: "128Mi"
       cpu: "500m"
       #Port to expose in the container
       ports:
       - containerPort: 80
```

1. To run the pod, run:
    a. `kubectl create -f pod.yaml`
2. To find out whether the pod is running successfully, you can run:
    a. `kubectl get all` (This command fetches all the resources in the default namespace)
    b. `kubectl get pods`(This command shows only the pods in the default namespace)
3. The pod we are running is a web server, and we must be able to connect to the server.
    a. Since we have only a single pod, we want to just access THAT pod, and not a deployment (as we will see in the next task).
    b. Run. `kubectl port-forward pod/nginx 80:80`
        i. What this command does is, say "I have a pod called Nginx, expose it such that if any HTTP request is coming for port 80 of the host machine, redirect it to me(Nginx pod), I (pod) will handle that request"
        ii. Note: You may see "Unable to Create Listener" errors if port 80 is already being used on the host system. To resolve this, you can use a different port. In the above command, the following is the format: <port_on_host>:<default_container_port>. The default container port here is the default Nginx port i.e 80. Therefore to expose another port say 8080, use 8080:80 . This is important to note every time you try to port-forward.
    c. Access your webserver through http://localhost:80 (Or the port you changed to , if port 80 was not available)
    d. Press Ctrl+C to stop the port-forwarding.
4. Delete the pods by either:
    a. Deleting every pod in the default namespace, by running:
        i. `kubectl delete pods --all`
    b. Deleting the specific pod, by :
        i. Finding the pod name, using:
            1. `kubectl get pods`
        ii. Deleting the specific pod by running
            1. `kubectl delete pod/<pod_name>`

TASK-3 (Create a Kubernetes Deployment and create a Load Balancing Service)

A pod is only a single instance(the smallest part) in a Kubernetes "deployment". A Kubernetes deployment refers to a collection of pods called replica sets, and configuration parameters for the replica sets. Deployment is used to tell Kubernetes how

to create or modify instances of the pods that hold a containerized application. Deployments can scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary. A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them. What this means is, it defines an abstract layer on top of a logical set of pods, essentially a “micro-service”-layer. A microservice can be complex and also be horizontally scaled, as a user/caller of the micro-service, I am not bothered by which specific pod services my request but I only care of the “service”, this is essentially what kubernetes service does. There are many types of Kubernetes services such as LoadBalancer, NodePort, Ingress etc. This task involves creating a deployment and then a service for it.

Contents of deploy.yaml

```
apiVersion: apps/v1
#Specifies the kind of resource we are creating
kind: Deployment
#Metadata to identify the deployment in the cluster
metadata:
  name: mynginx
#Specifications of the deployment
spec:
  #The number of replicas the deployment must have
  replicas: 2
  #Metadata to identify the replicaset in the cluster
  selector:
       matchLabels:
       app: mynginx
  template:
       metadata:
       labels:
       app: mynginx
       # Pod specs
       spec:
       containers:
       - name: mynginx
       image: nginx:latest
       resources:
```

```
      limits:
            memory: "128Mi"
            cpu: "500m"
      ports:
      - containerPort: 80
```

Contents of deploy_service.yaml

```
apiVersion: apps/v1
#Specifies the kind of resource we are creating
kind: Deployment
#Metadata to identify the deployment in the cluster
metadata:
  name: mynginx
#Specifications of the deployment
spec:
  #The number of replicas the deployment must have
  replicas: 2
  #Metadata to identify the replicaset in the cluster
  selector:
      matchLabels:
      app: mynginx
  template:
      metadata:
      labels:
      app: mynginx
      # Pod specs
      spec:
      containers:
      - name: mynginx
      image: nginx:latest
      resources:
      limits:
            memory: "128Mi"
            cpu: "500m"
      ports:
      - containerPort: 80
```

```
# This divider is a YAML specification to seperate different YAML scripts in the
same file

---



apiVersion: v1

#Specifies the kind of resource we are creating

kind: Service

metadata:

  name: nginx

spec:

# This selector is IMPORTANT, it selects those pods in the deployment that match
the below key-value pair

# Only those pods that match get traffic from the service

  selector:

       app: mynginx

  ports:

  # The type of protocol the service exposes

       - protocol: TCP

  # What is the port the SERVICE must expose(This field is not needed for
NodePort services)

       port: 80

  # What is the port that the POD EXPOSES

       targetPort: 80

# The type of service we are creating

  type: LoadBalancer
```

1. Download the deploy.yaml file which defines the deployment. Go through the deployment YAML file to understand what each field is and how it affects the deployment. **Ref links to understand resource limits used in the YAML file and their meaning :** https://jamesdefabia.github.io/docs/user-guide/compute-resources/ or https://medium.com/@betz.mark/understanding-resource-limits-in-kubernetes-cpu-time-9eff74d3161b
2. Just like a pod, run the deployment using the following command:

   ```
   kubectl create -f deploy.yaml
   ```

3. This deployment creates 2 pods of NGINX web server and a replica set to manage these pods. To view all the aspects of the deployment(pods, replica sets etc), run:

   ```
   kubectl get all
   ```

4. Once you see all the pods up and running, your deployment is complete. The newly created replica set should show 2/2.
5. To create a service for this deployment, we can either use:
   a. `kubectl expose deploy mynginx --port=80 --target-port=80`
   b. Or use a YAML specification which **we will do now**.
6. Delete the previous deployment and service by running:

   ```
   kubectl  delete deploy mynginx
   ```

   ```
   kubectl delete service mynginx
   ```

7. Download the deploy_service.yaml file.
8. Create the deployment **and its service** by running:

   ```
   kubectl create -f deploy_service.yaml
   ```

9. Once the pods are up and running, expose the service using:

   ```
   kubectl port-forward service/nginx 80:80
   ```

10. Access the webserver on http://localhost:80
11. Keep the deployment and service running for the next task

TASK-4 (Scaling deployments)

Replica sets are what control the pods in a deployment, they are what allow for creating rolling updates, config changes etc but more importantly scaling. Replica sets scale the pods in the deployment without affecting the micro-service availability.

1. To scale the deployment done before, run the following command:

   ```
   kubectl scale deploy mynginx --replicas=10
   ```

2. The above command scales the pods to 10 pods. (You may see pods in a pending state, this is because we are using minikube which has resource restrictions, on a real cluster having sufficient resources we would see the scaling successfully done)
3. Delete the deployment and the service:

   a. `kubectl delete service nginx`

   b. `kubectl delete deploy mynginx`

TASK-5 (Shutdown minikube)

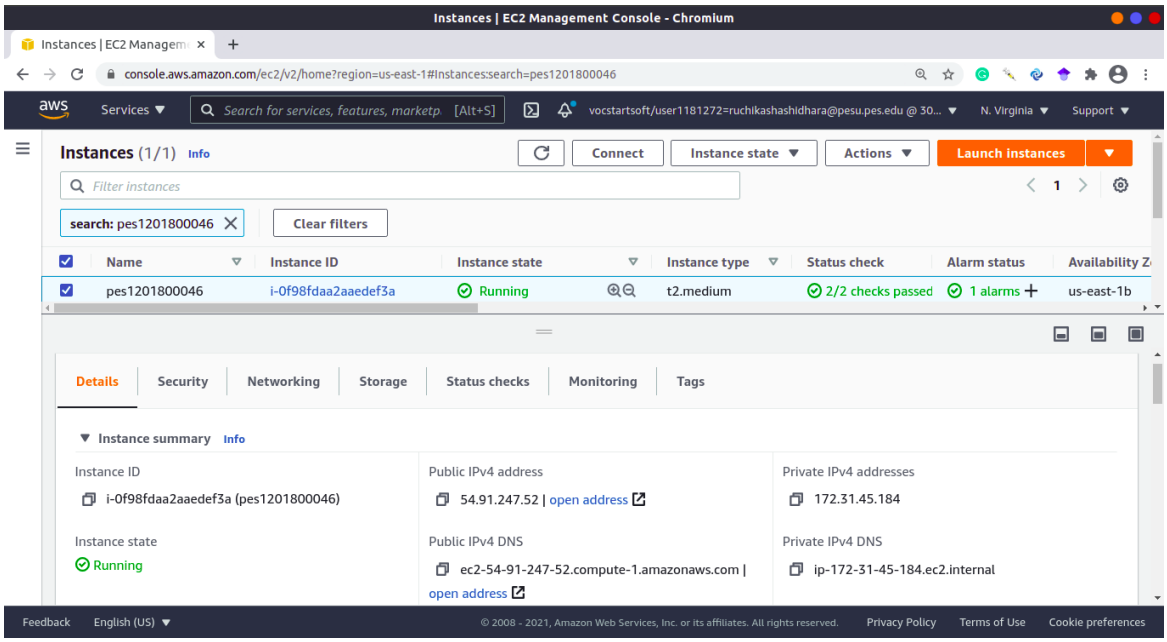
This task ensures all resources are stopped:

1. Ensure all resources are stopped and not seen when running the Kubernetes commands.
2. Stop minikube by running:

   ```
   minikube stop
   ```

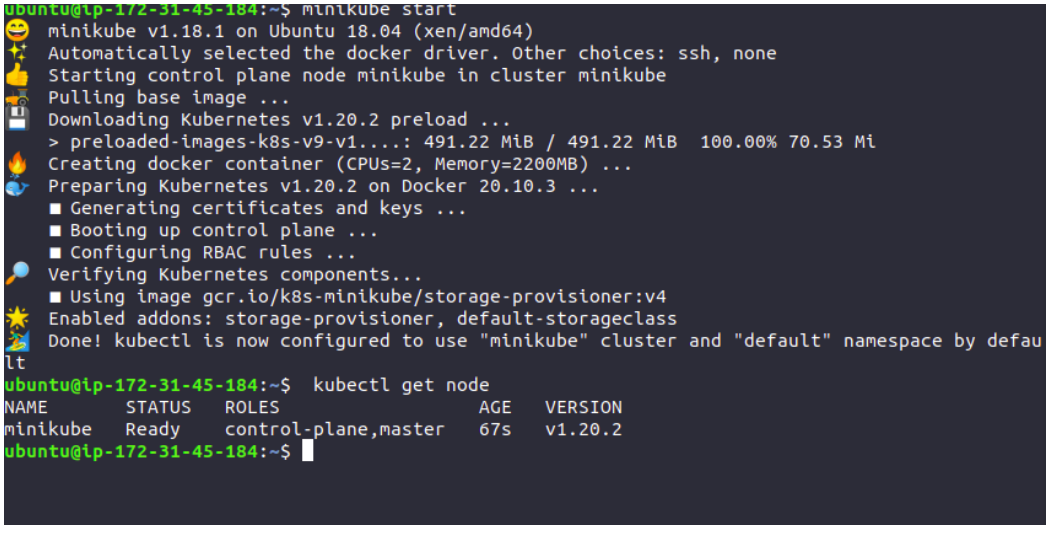
LAB 8 SCREENSHOTS

Task 0: EC2 Instance Configuration along with its Public & Private IPv4, DNS addresses



Task 1: Create a Kubernetes cluster using minikube

1a.png Kubectl get nodes showing minikube is running properly



Task 2: Kubernetes pods

2a.png Screenshot showing:

nginx pod in running status

port-forwarding of the pod using port 80 & 8010

```
ubuntu@ip-172-31-45-184:~/task2$ kubectl get all
NAME        READY   STATUS    RESTARTS   AGE
pod/nginx   1/1     Running   0          8s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   63m
ubuntu@ip-172-31-45-184:~/task2$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          15s
ubuntu@ip-172-31-45-184:~/task2$ kubectl port-forward pod/nginx 8010:80
Forwarding from 127.0.0.1:8010 -> 80
Forwarding from [::1]:8010 -> 80
Handling connection for 8010
```

2b.png Nginx home page access through localhost

Task 3: Kubernetes Deployments and Kubernetes services

3a.png Deployment running and replicas as 2/2 along withservice

```
ubuntu@ip-172-31-45-184:~/task3$ kubectl create -f deploy_services.yaml
deployment.apps/mynginx created
service/nginx created
ubuntu@ip-172-31-45-184:~/task3$ kubectl get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/mynginx-88bb55cb6-lx5cp     1/1     Running   0          9s
pod/mynginx-88bb55cb6-xpl5w     1/1     Running   0          9s

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP        8h
service/nginx        LoadBalancer   10.110.3.103    <pending>     80:31543/TCP   9s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mynginx    2/2     2            2           9s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/mynginx-88bb55cb6     2         2         2       9s
ubuntu@ip-172-31-45-184:~/task3$ kubectl port-forward service/nginx 8050:80
Forwarding from 127.0.0.1:8050 -> 80
Forwarding from [::1]:8050 -> 80
Handling connection for 8050
```

3b.png Port-forwarding of the service using port 80 and accessing the web page using localhost

Task 4: Scaling Kubernetes

4a.png showing 10 pods (either running/pending state)

```
ubuntu@ip-172-31-45-184:~/task3$ kubectl scale deploy mynginx --replicas=10
deployment.apps/mynginx scaled
ubuntu@ip-172-31-45-184:~/task3$ kubectl get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/mynginx-88bb55cb6-2cfdm     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-97j58     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-bxqm9     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-f4wld     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-jcrn9     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-lx5cp     1/1     Running   0          5m58s
pod/mynginx-88bb55cb6-v4s6d     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-wnf92     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-xfw89     0/1     Pending   0          12s
pod/mynginx-88bb55cb6-xpl5w     1/1     Running   0          5m58s

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP        8h
service/nginx        LoadBalancer   10.110.3.103    <pending>     80:31543/TCP   5m58s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mynginx    2/10    10           2           5m58s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/mynginx-88bb55cb6     10        10        2       5m58s
```

Lab 9 - AWS EMR

Introduction:

Welcome to Lab 9! This simple exercise is designed to introduce you to AWS EMR

Reading - 30 mins

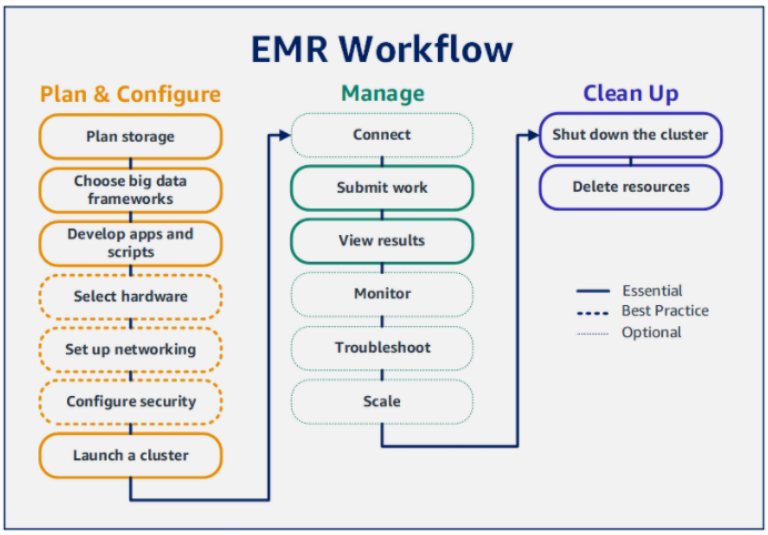
What is AWS EMR?

Amazon EMR is the industry-leading cloud big data platform for processing vast amounts of data using open source tools such as Apache Spark, Apache Hive, Apache HBase, Apache Flink, Apache Hudi, and Presto. Amazon EMR makes it easy to set up, operate, and scale your big data environments by automating time-consuming tasks like provisioning capacity and tuning clusters. With EMR you can run petabyte-scale analysis at less than half of the cost of traditional on-premises solutions and over 3x faster than standard Apache Spark. You can run workloads on Amazon EC2 instances, on Amazon Elastic Kubernetes Service (EKS) clusters, or on-premises using EMR on AWS Outposts.

Overview of AWS EMR :
https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html

Read about Spark : https://spark.apache.org/

EMR Workflow



Reference:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html

TASK 1: Plan and Configure an Amazon EMR Cluster

Name of s3 bucket: [your-srn-bucket]

Upload both .csv and .py files into S3 bucket and take a screenshot with uploaded files required (9a)

Launching cluster

Cluster name : my-cluster-[yoursrn]

Use the following configurations in your cluster:

Release label: emr-6.2.0

Application: Spark

Instance type: m5.xlarge

Instance count: 3 (1 master and 2 workers)

Screenshot of running cluster (9b)

TASK2: Managing Amazon EMR Clusters

Screenshot of spark job completed (9c)

Screenshot of output folder in S3 (9d)

Download the output csv to your local machine and upload csv (and NOT the screenshot of the csv) to the drive (9e.csv)

NOTE: In case "Add step" task in Cluster Management fails, make sure you have added both .csv file and .py file in your S3 bucket and then repeat the "Add step" task. At this point, you can add a new step or clone the existing (failed) step but make sure all the configurations are correct for the spark application before you re-run the step.

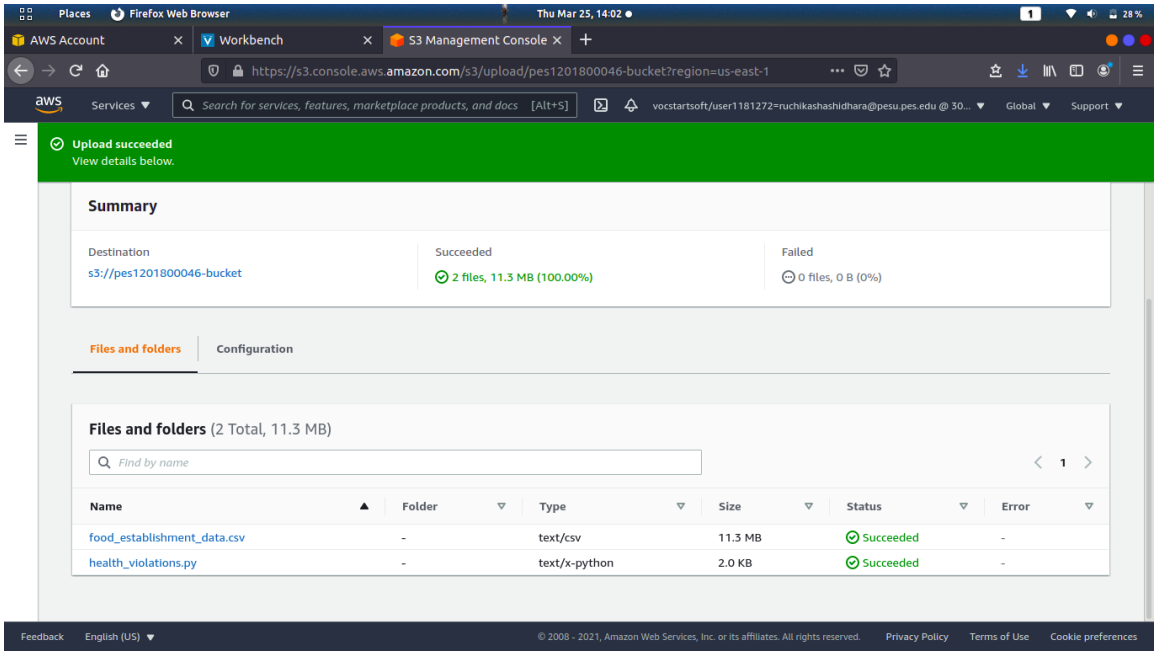
TASK3: Clean Up Amazon EMR Cluster Resources (Do not forget)

Go to EMR, select your cluster and click "Terminate".

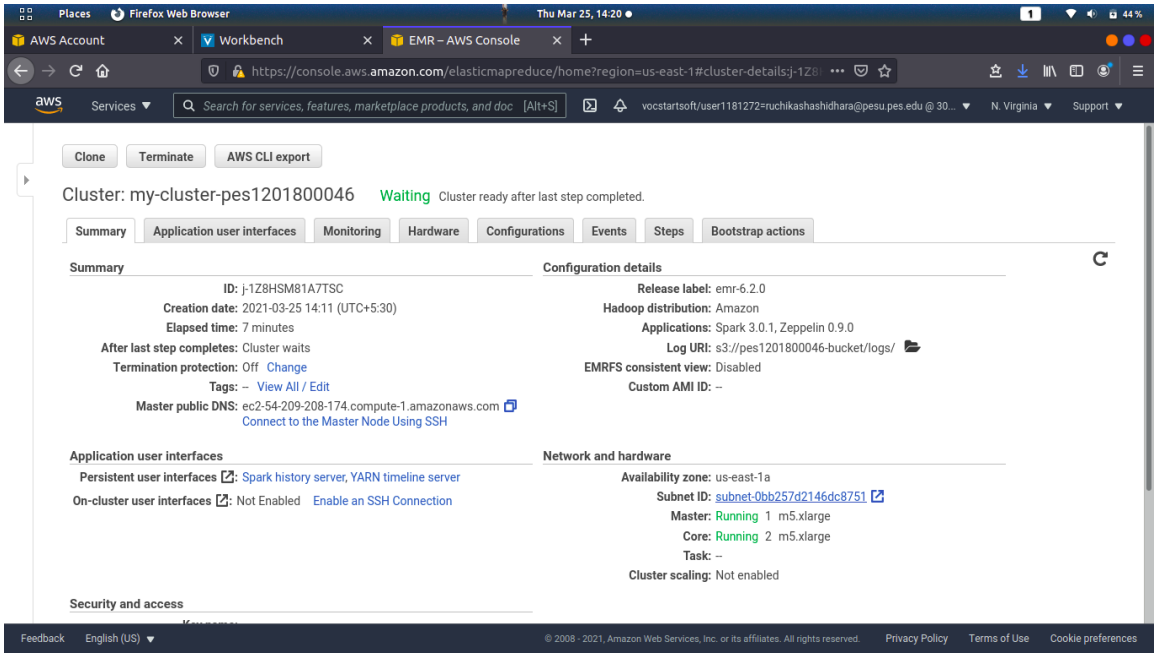
LAB 9 SCREENSHOTS

Task 1: Plan and Configure an Amazon EMR Cluster

9a.png - Screenshot of food_establishment_data.csv & health_violations.py uploaded into the S3 Bucket

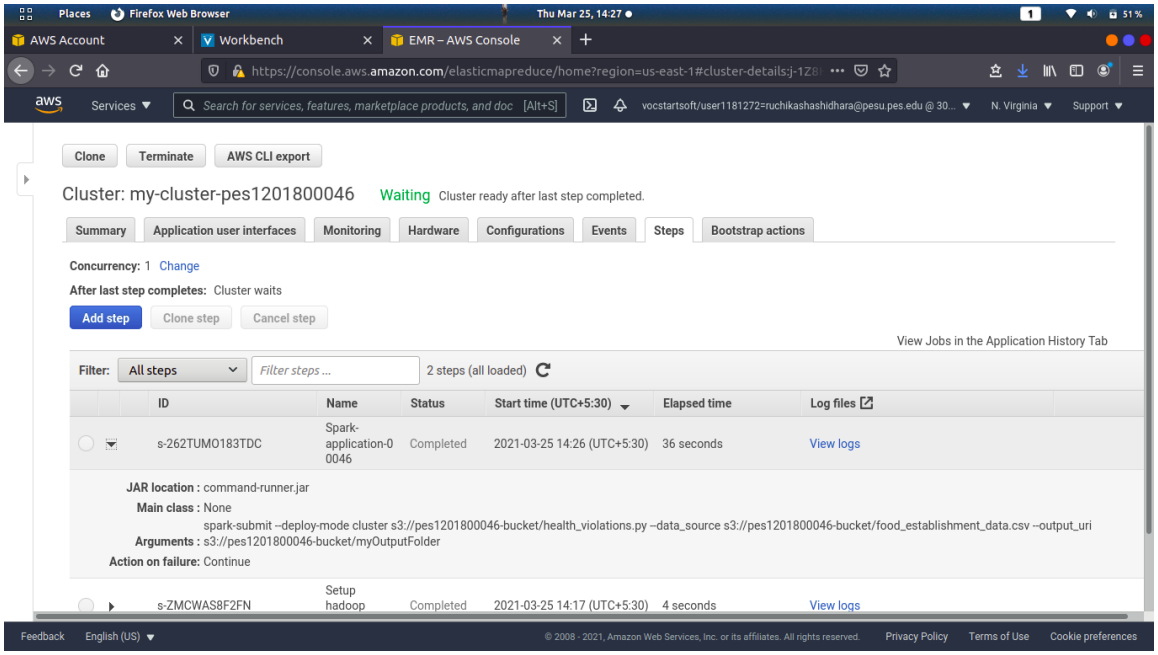


9b.png - Screenshot of the running cluster

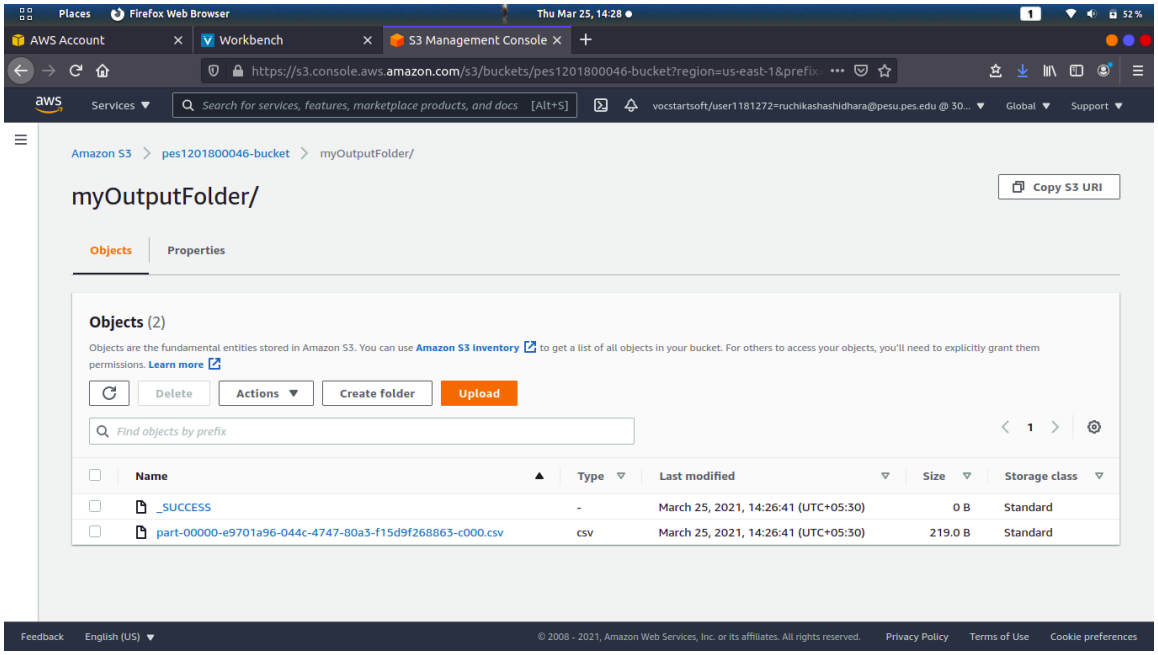


Task 2: Managing Amazon EMR Cluster

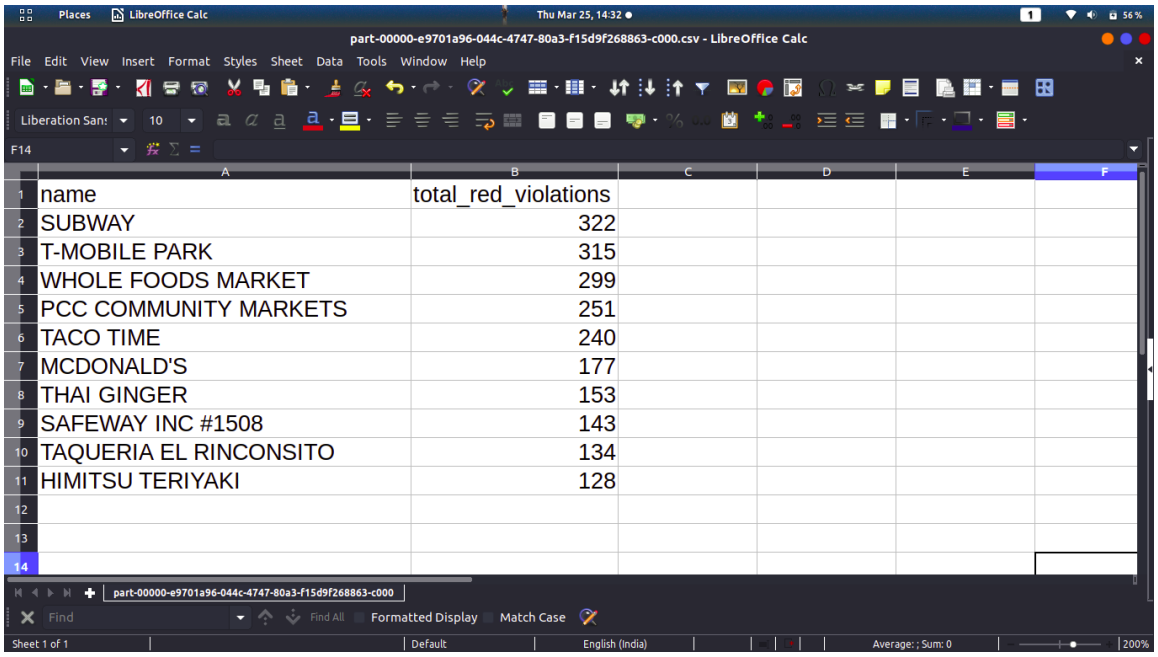
9c.png - Screenshot of the Completed Spark Job



9d.png - Screenshot of the output folder in the S3 Bucket



9e.png - Screenshot of the output .csv file



	A	B	C	D	E	F
1	name	total_red_violations				
2	SUBWAY	322				
3	T-MOBILE PARK	315				
4	WHOLE FOODS MARKET	299				
5	PCC COMMUNITY MARKETS	251				
6	TACO TIME	240				
7	MCDONALD'S	177				
8	THAI GINGER	153				
9	SAFEWAY INC #1508	143				
10	TAQUERIA EL RINCONSITO	134				
11	HIMITSU TERIYAKI	128				
12						
13						
14						

Lab 10 - Serverless Computing With AWS Lambda & AWS API Gateway

Introduction to Qwiklabs

Qwiklabs is an online platform that provides end to end training in Cloud Services. This a platform where you can learn in a live environment anywhere, anytime and on any device. Qwiklabs offers training through various Labs which are specially designed to get you trained in Google Cloud Platform (GCP) as well as Amazon Web Services (AWS). Qwiklabs has joined hands with Google and now works as a part of Google Cloud. Every year Qwiklabs delivers thousands of labs and has happy learners all over the globe.

Introduction to Serverless Computing

Serverless computing is a method of providing backend services on an as-used basis. A serverless provider allows users to write and deploy code without the hassle of worrying about the underlying infrastructure. A company that gets backend services from a serverless vendor is charged based on their computation and do not have to reserve and pay for a fixed amount of bandwidth or number of servers, as the service is auto-scaling. Note that despite the name serverless, physical servers are still used but developers do not need to be aware of them.

What is serverless computing? | Serverless definition

What is Serverless?

Serverless Computing – Amazon Web Services

AWS Lambda Serverless Functions

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration.

Introduction to AWS Lambda - Serverless Compute on Amazon Web Services

In this lab task, you will be creating a serverless lambda function to automatically create thumbnails. Whenever an image is uploaded to the S3 bucket, it should automatically trigger the lambda function.

TASK1: Click on the below link to go to Qwiklabs lab: Introduction to AWS Lambda - Qwiklabs Lab and Click "Join to Start This Lab"

AWS API Gateway

AWS API gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor and secure APIs. API acts as a front door for the application to access data, business logic or functionality from the backend services. It handles all the task involved in accepting and processing up of hundreds or thousands of concurrent API calls, including traffic management, authorization, access control, monitoring and API management.

https://aws.amazon.com/api-gateway/

Building APIs with Amazon API Gateway

Five Reasons to Consider Amazon API Gateway for Your Next Microservices Project – The New Stack

This lab task will involve using your learnings from the previous task(Lambda). In this lab task you will create an AWS lambda function to return random Questions and answers(FAQs), and this lambda function will be exposed using a REST API endpoint created(and managed) using API Gateway.

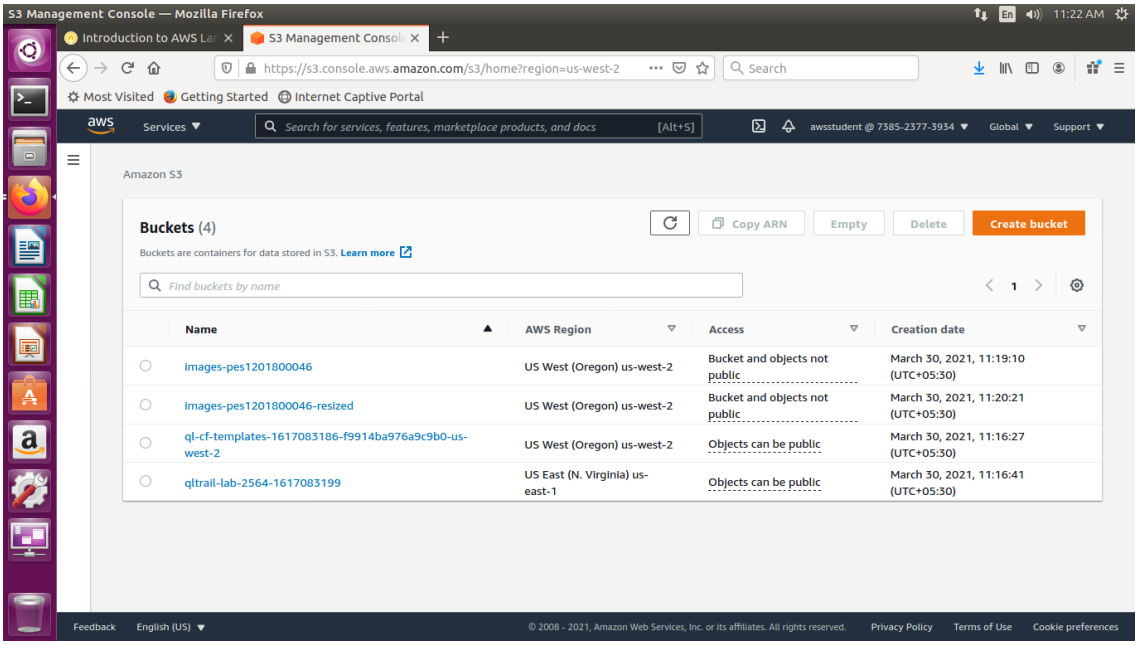
Point to keep in mind: While in AWS Lambda console, ensure that you have UNCHECKED using the new UI, make sure to use the old/former UI as the Qwiklabs have not yet been updated for the new UI

TASK2: Click on the below link to go to Qwiklabs lab: Introduction to AWS API Gateway- Qwiklabs Lab and Click "Join to Start This Lab"

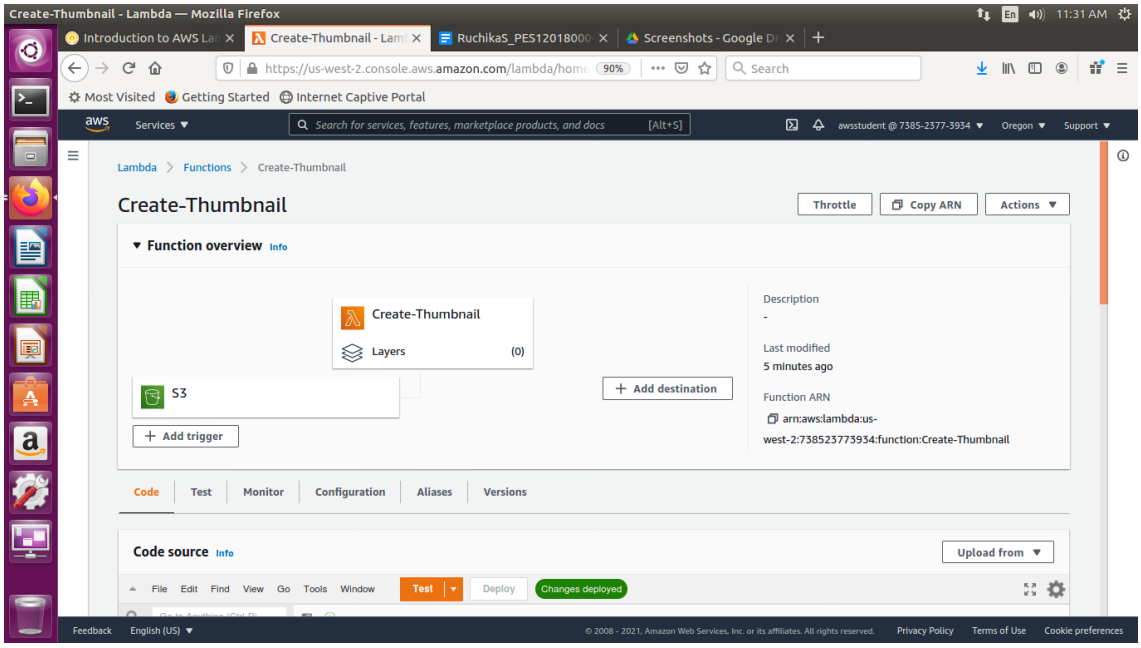
LAB 10 SCREENSHOTS

Task 1 - AWS Lambda

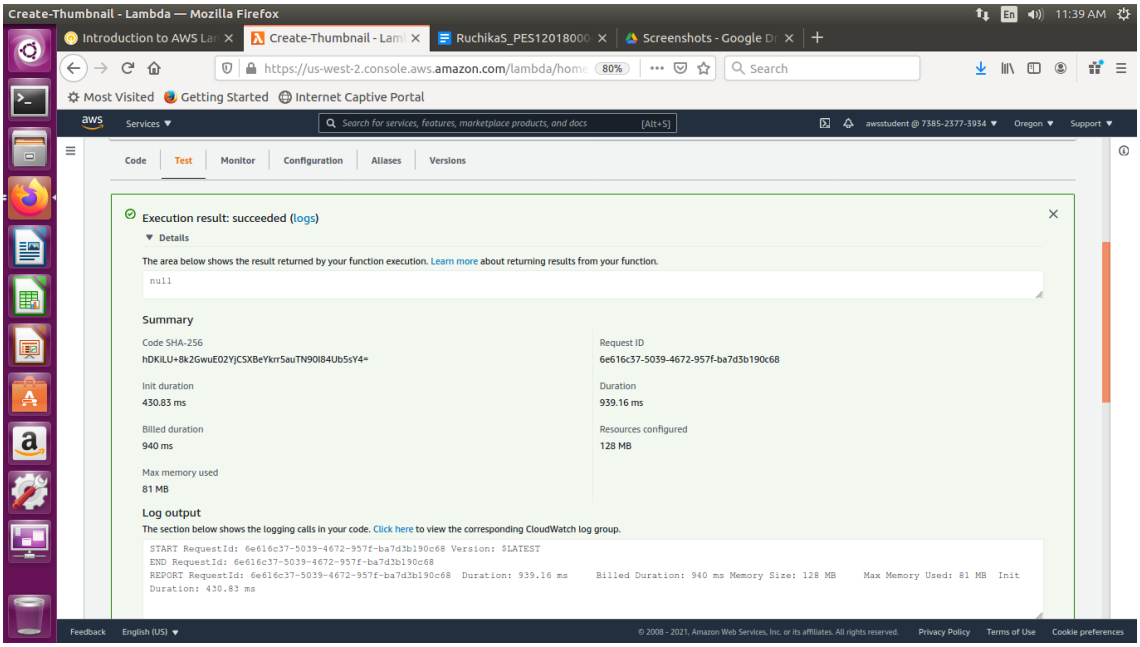
1a.png - Showing 2 S3 buckets created, one having the original image and another S3 bucket suffixed with - resized to store thumbnails



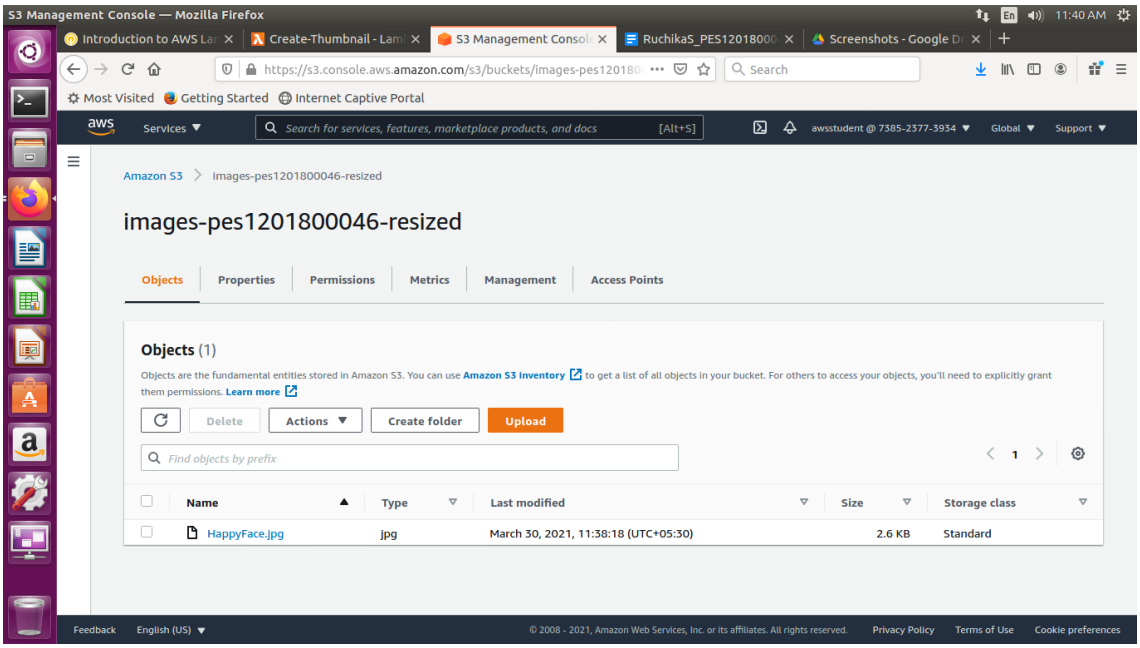
1b.png - Showing the Designer tab diagram, showing that S3 will trigger the creation of the create-thumbnail function



1c.png - Test run successfully showing test details also

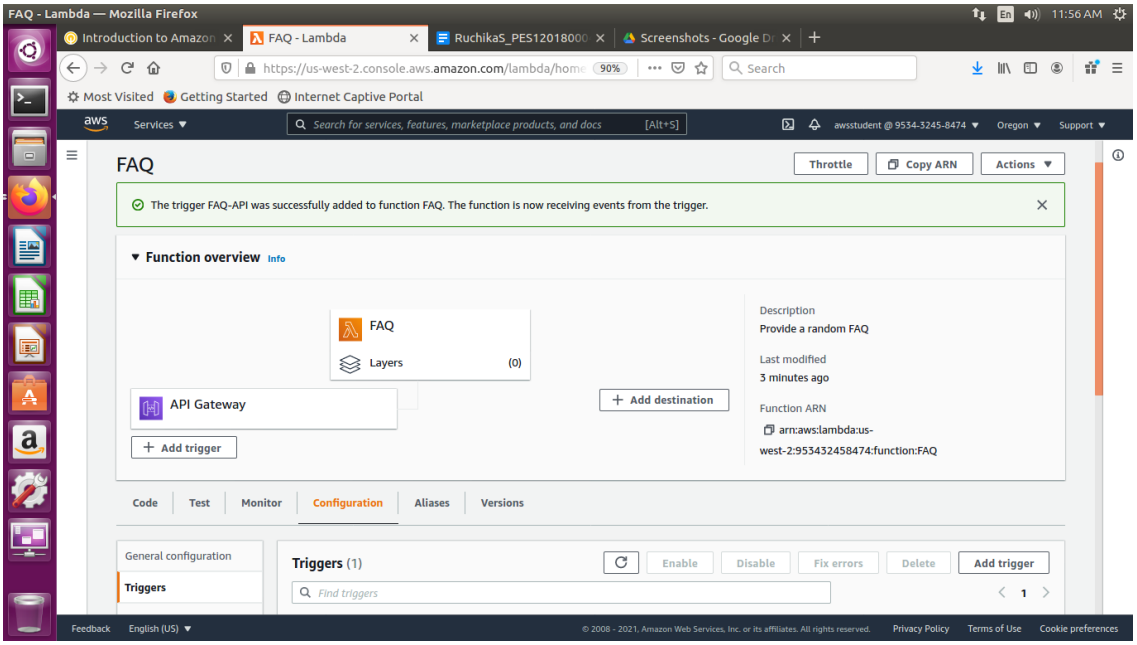


1d.png - Thumbnail created in the *-resized* S3 bucket

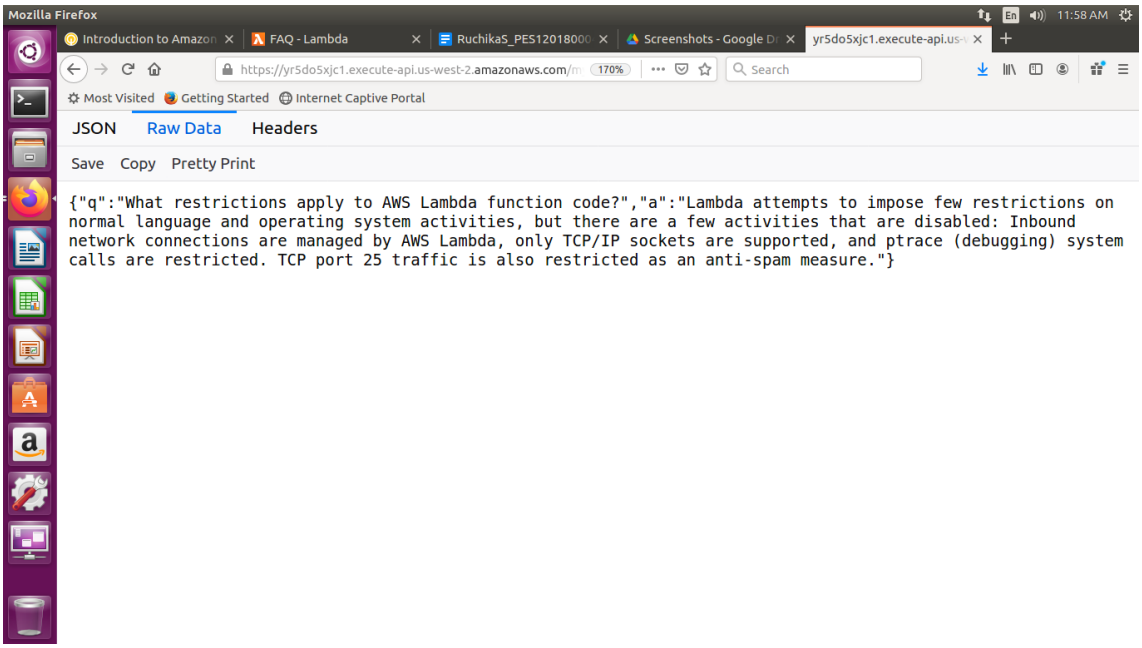


Task 2 - API Gateway with AWS Lambda

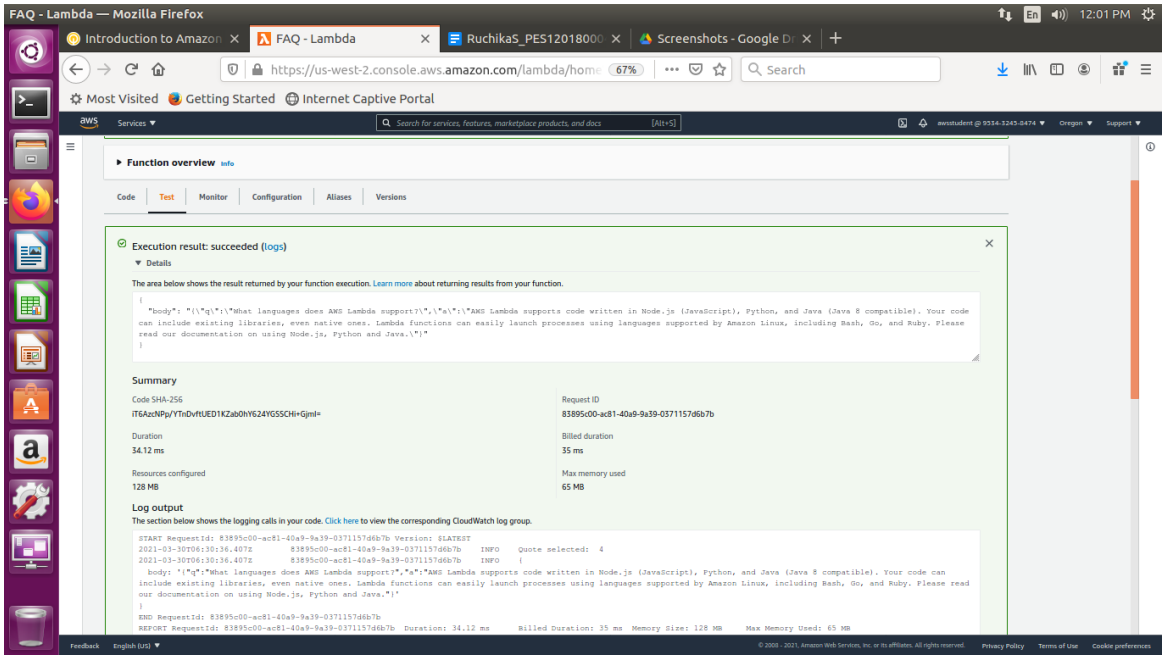
2a.png - Showing the Designer tab diagram, showing that API Gateway will trigger the creation of the FAQ function



2b.png - Screenshot of accessing the API Endpoint URL on browser and getting successful response



2c.png - Test run successfully showimng test logs also



Lab 11 - Introduction to AWS Identity and Access Management (IAM)

Introduction to Qwiklabs

Qwiklabs is an online platform that provides end to end training in Cloud Services. This a platform where you can learn in a live environment anywhere, anytime and on any device. Qwiklabs offers training through various Labs which are specially designed to get you trained in Google Cloud Platform (GCP) as well as Amazon Web Services (AWS). Qwiklabs has joined hands with Google and now works as a part of Google Cloud. Every year Qwiklabs delivers thousands of labs and has happy learners all over the globe.

Introduction to IAM

AWS Identity and Access Management (IAM) enables you to manage access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

IAM is a feature of your AWS account offered at no additional charge. You will be charged only for use of other AWS services by your users.

How it works?

IAM assists in creating roles and permissions. AWS IAM allows you to:

Manage IAM users and their access – You can create users in IAM, assign them individual security credentials (in other words, access keys, passwords, and multi-factor authentication devices), or request temporary security credentials to provide users access to AWS services and resources. You can manage permissions in order to control which operations a user can perform.

Manage IAM roles and their permissions – You can create roles in IAM and manage permissions to control which operations can be performed by the entity, or AWS service, that assumes the role. You can also define which entity is allowed to assume the role. In addition, you can use service-linked roles to delegate permissions to AWS services that create and manage AWS resources on your behalf.

Manage federated users and their permissions – You can enable identity federation to allow existing identities (users, groups, and roles) in your enterprise to access the AWS Management Console, call AWS APIs, and access resources, without the need to create an IAM user for each identity. Use any identity management solution that supports SAML 2.0, or use one of our federation samples (AWS Console SSO or API federation).

Click on the link below to go to Qwiklabs lab (and then click **Join to Start this Lab** followed by **Start Lab** after which you will see **Open Console** button appearing on the left panel after 1 to 2 minutes) **AWS IAM**

Tasks to be performed:

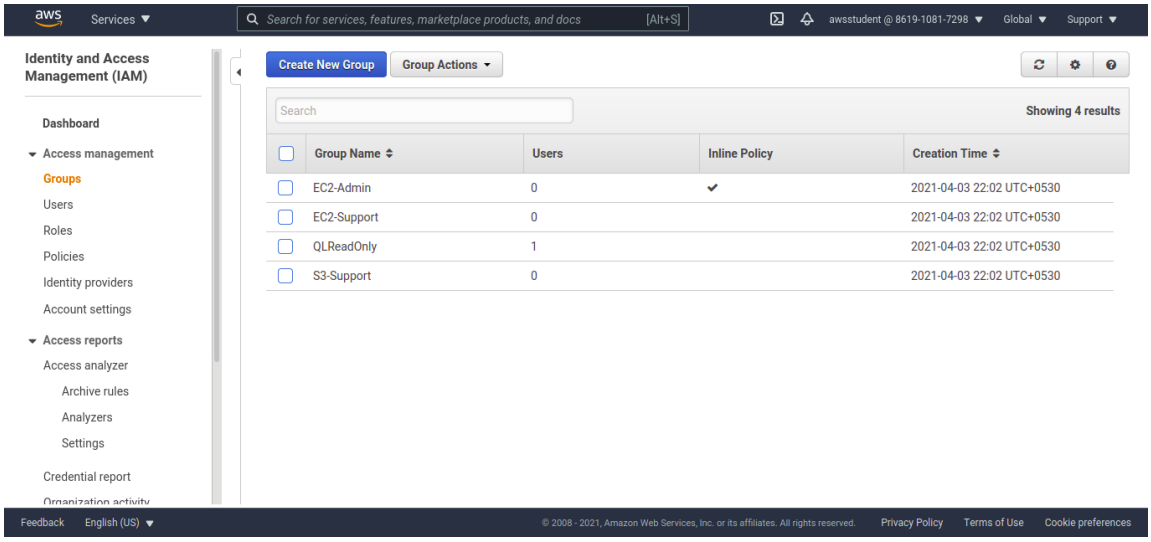
Task-1: Explore the Users and Groups (You need to execute Steps 3-21 here)
Task-2: Add Users to Groups (Execute Steps 22-29)
Task-3: Sign-In and Test Users (Execute Steps 30-59)
Task-4: End Lab (Execute Steps 60-64)

During Task-3 you will be asked to logout and log in with a different user ID three times, so note down the Account ID and the user sign-in link (which will look similar to https://123456789012.signin.aws.amazon.com/console) displayed on IAM dashboard in a separate text file. In this link 123456789012 denotes the Account ID which you will need to login with a different user id.

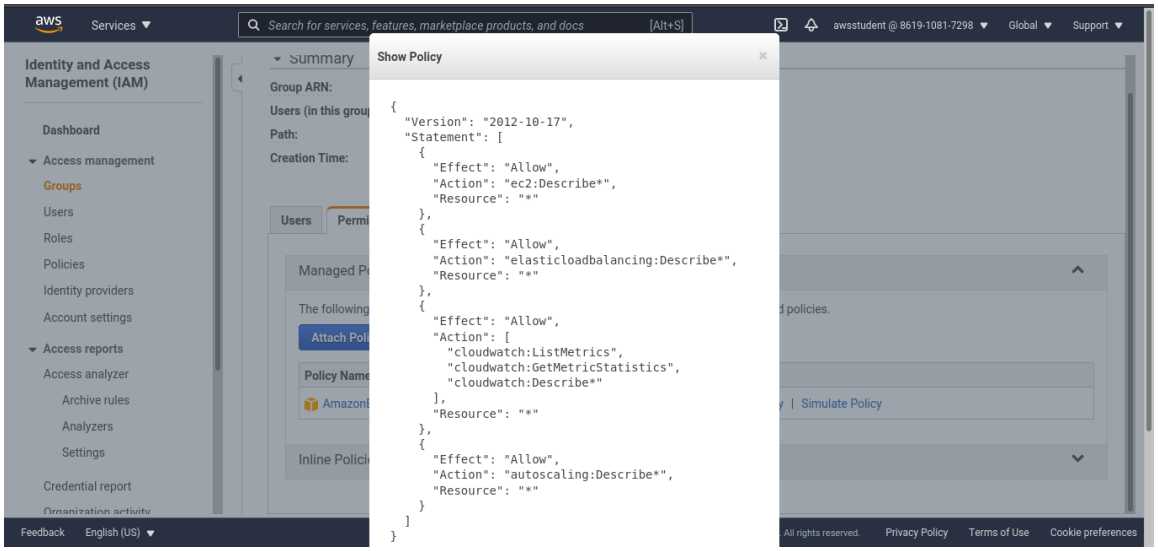
LAB 11 SCREENSHOTS

Task 1: Explore the Users and Groups

1a.png - List of groups showing that they have zero users in each group

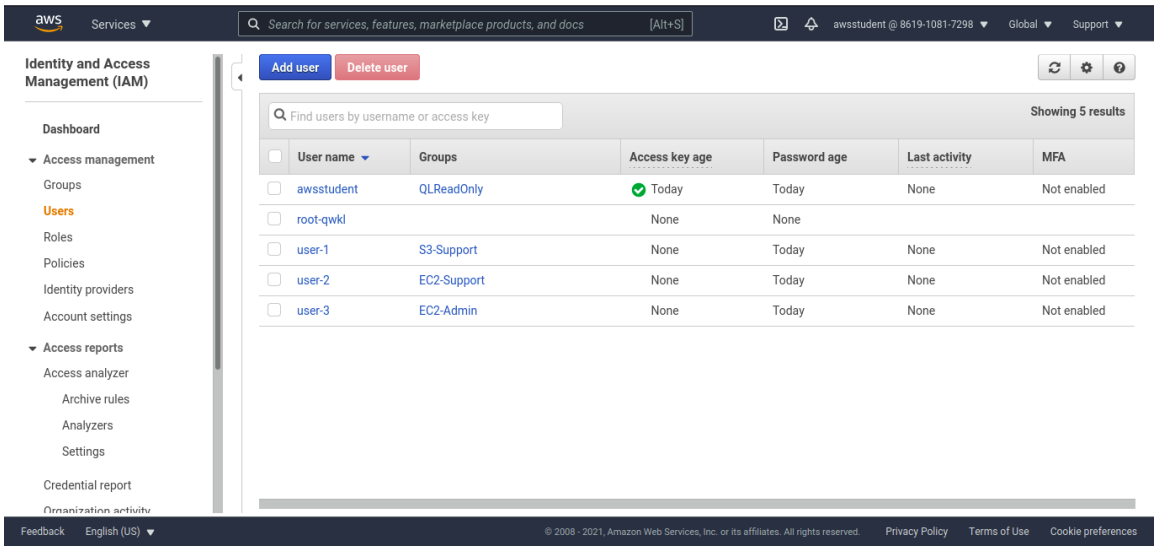


1b.png - Showing the policy for EC2-Support group

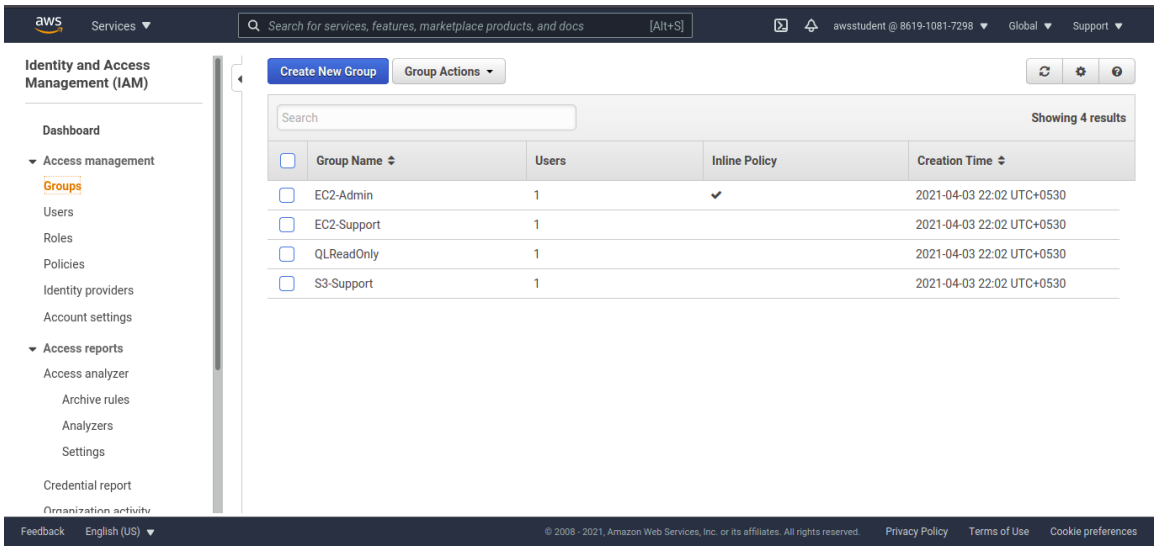


Task 2: Add Users to the Groups

2a.png - Showing each user with the groups they have been added to

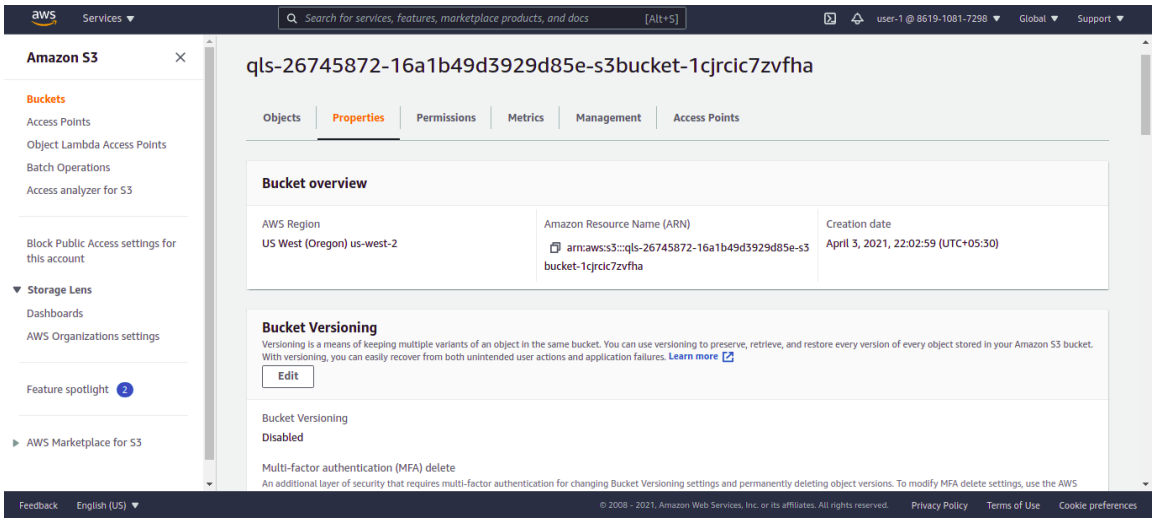


2b.png - Showing each group having one user

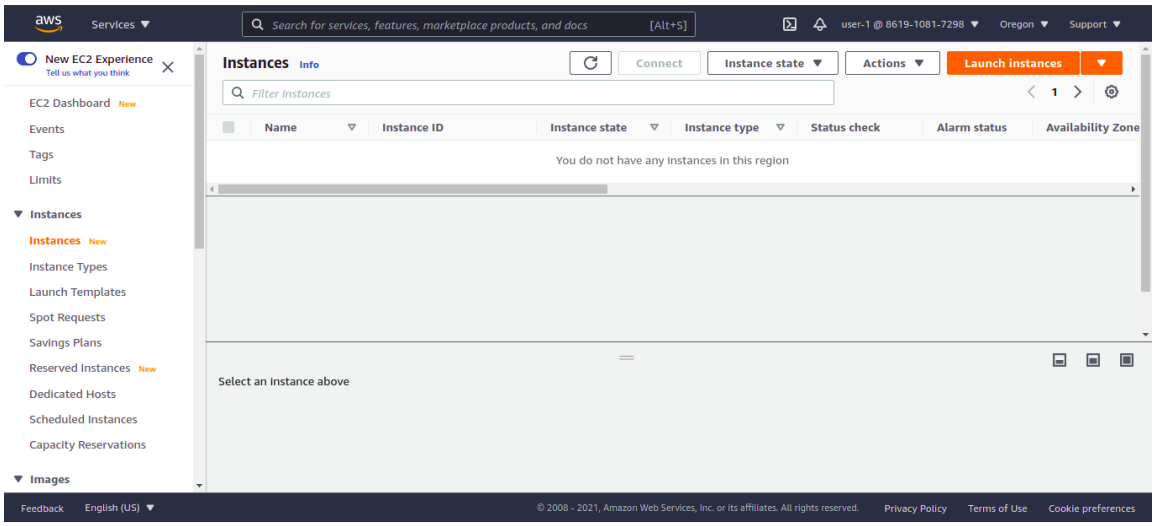


Task 3: Testing User Access

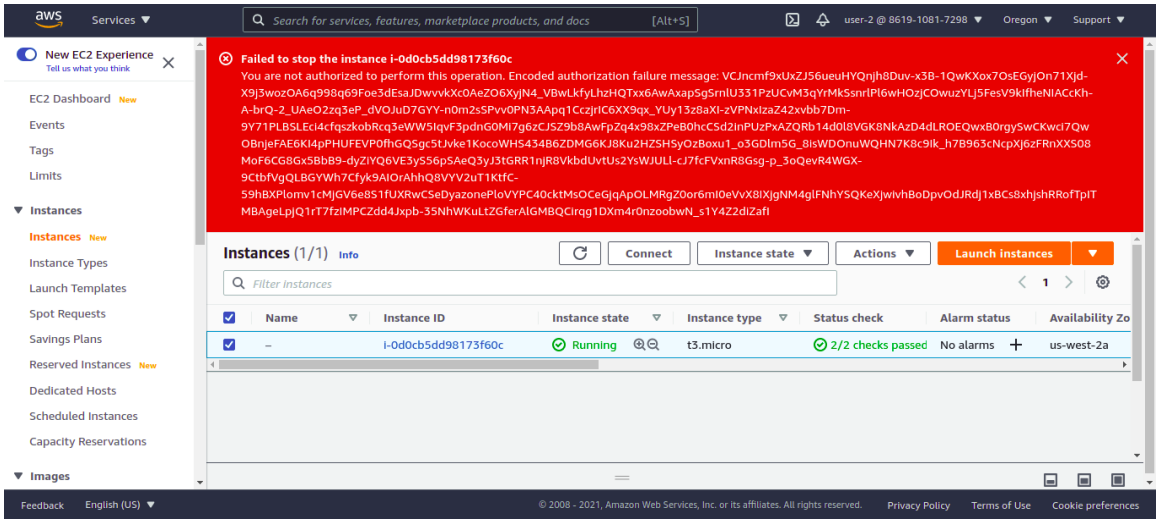
3a.png - Showing user-1 having access to view details of the S3-Bucket



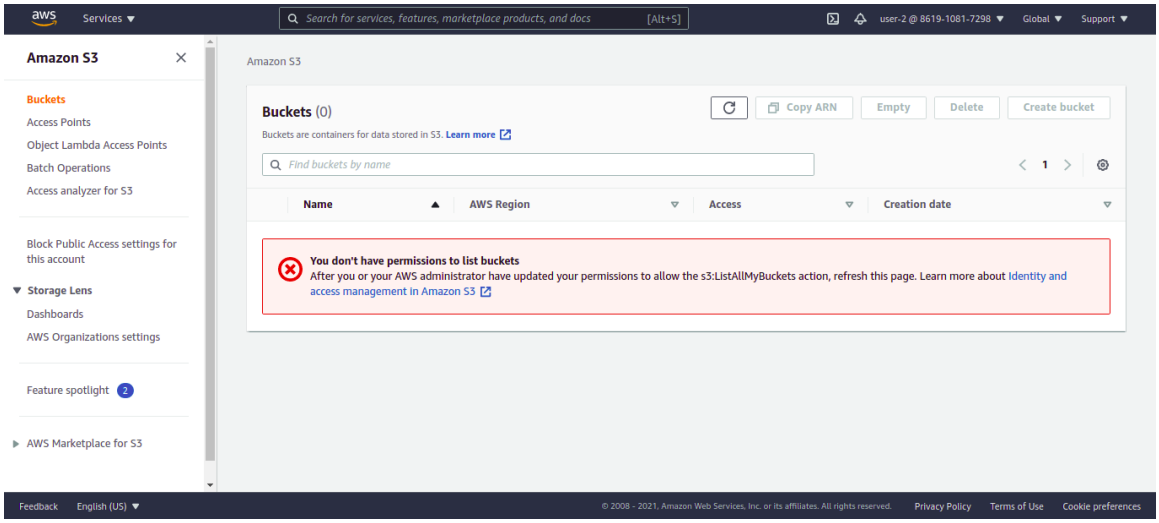
3b.png - Showing user-1 having NO access to view details of the EC-2 Instances



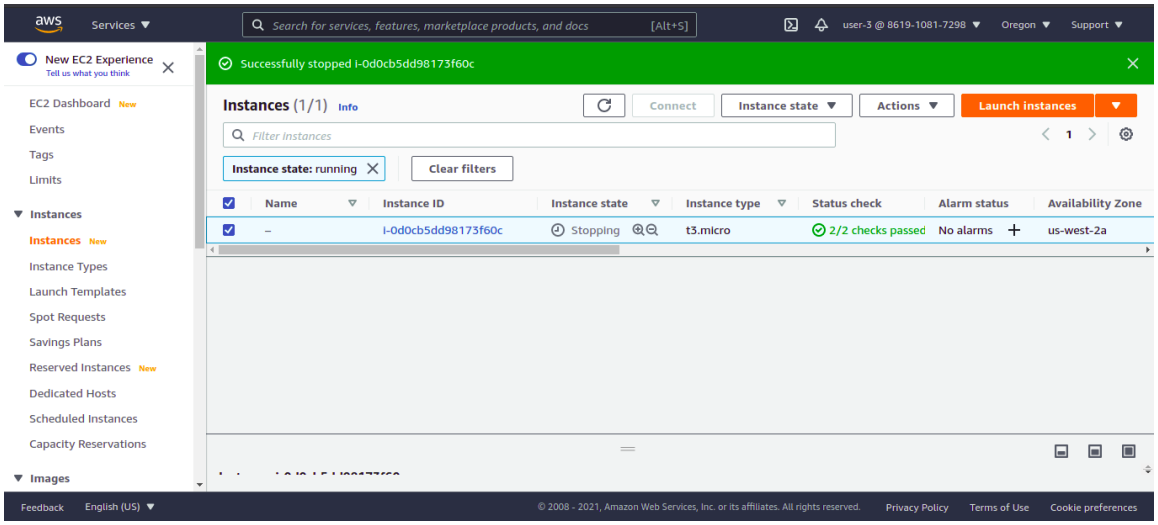
3c.png - Showing user-2 having access view details of the EC-2 Instances, but not able to stop instance



3d.png - Showing user-2 having NO access to view details of S3-Buckets



3e.png - Showing user-3 having able to stop instance, since the user is a admin



Task 4: Ending the Lab

4a.png - Showing email confirmation from QWIKLABS for having completed the lab successfully, clearly showing the timestamp in the received email