

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
chmod 400 keynote.pem
ssh -i keynote.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-system

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:

- Give the description as: "Snapshot of my volume"
- Key: "Name"
- 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)

The screenshot shows the 'Launch Instance wizard' in the EC2 Management Console. The current step is '7. Review'. The page displays the configuration for launching an Ubuntu Server 20.04 LTS (HVM) instance. Key details include:

- AMI:** Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-0885b1f6bd170450c
- Instance Type:** t2.micro
- Root Device Type:** ebs
- Virtualization type:** hvm
- Security Groups:** launch-wizard-2
- Description:** launch-wizard-2 created 2021-01-22T00:20:12.183+05:30
- Ports:** SSH (TCP, Port 22, Source 0.0.0.0/0)

At the bottom, there are 'Cancel', 'Previous', and 'Launch' buttons.

1b2.png showing the EC2 instance in running state

The screenshot shows the 'Instances' page in the EC2 Management Console. A blue banner at the top says 'Welcome to the new instances experience!'. The main table shows one instance:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
PES1201800...	i-064c814e00ce575a3	Running	t2.micro	2/2 checks ...	1 alarm ...	us-east-1e	ec2-3-85-4...

Below the table, the 'Instance summary' section provides detailed information for the instance i-064c814e00ce575a3 (PES1201800046). It includes fields for:

- Instance ID: i-064c814e00ce575a3 (PES1201800046)
- Public IPv4 address: 3.85.41.128 | open address
- Private IPv4 address: 172.31.57.74
- Public IPv4 DNS: ec2-3-85-41-128.compute-1.amazonaws.com | open address
- Private IPv4 DNS: ip-172-31-57-74.ec2.internal

At the bottom, there are 'Details', 'Security', 'Networking', 'Storage', 'Status Checks', 'Monitoring', and 'Tags' tabs.

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

```
Fri Jan 22, 00:36 ●
ubuntu@ip-172-31-57-74:~●
Places Terminal Fri Jan 22, 00:36 ●
ubuntu@ip-172-31-57-74:~●
File Edit View Search Terminal Help
ubuntu@ip-172-31-57-74:~$ ls
hello-world.c
ubuntu@ip-172-31-57-74:~$ cat hello-world.c
#include <stdio.h>
int main () {
    printf("C program on an AWS EC2 instance!\n");
    return 0;
}
ubuntu@ip-172-31-57-74:~$ gcc hello-world.c
ubuntu@ip-172-31-57-74:~$ ./a.out
C program on an AWS EC2 instance!
ubuntu@ip-172-31-57-74:~$
```

Task C:

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

The screenshot shows the AWS EC2 Management Console with the 'Volumes' page selected. The left sidebar includes 'New EC2 Experience', 'Events', 'Tags', 'Limits', 'Instances' (with 'Instances' selected), 'Images', 'AMIs', and 'Elastic Block Store' (with 'Volumes' selected). The main area displays a table of volumes, with one row highlighted. The highlighted row shows the following details:

Name	Volume ID	Size	Volume Type	IOPS	Throughput	Snapshot	Created	Availability Zone	State	Alarm Status	Attachment Information
vol-015bd34...	8 GiB	gp2	100	-	-	snap-0b071e0...	January 22, 2021 at...	us-east-1e	in-use	None	i-07e1bebe45d8407...
pes1201800...	vol-08000de...	10 GiB	gp2	100	-		January 22, 2021 at...	us-east-1e	available	None	

Below the table, a detailed view of the selected volume (vol-08000de15e30756d1) is shown. The 'Description' tab is active, displaying the following information:

Volume ID	vol-08000de15e30756d1
Alarm status	None
Snapshot	-
Availability Zone	us-east-1e
Encryption	Not Encrypted
KMS Key ID	-
KMS Key Aliases	-
KMS Key ARN	-
Throughput (MB/s)	-

On the right side, there is a summary table:

Outposts ARN	-
Size	10 GiB
Created	January 22, 2021 at 1:07:28 AM UTC+5:30
State	available

At the bottom, there are links for 'Feedback', 'English (US)', 'Privacy Policy', and 'Terms of Use'.

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.

The screenshot shows the AWS EC2 Management Console interface. On the left, a navigation sidebar lists various services: New EC2 Experience, Events, Tags, Limits, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images, AMIs, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, and Key Pairs. The main content area displays a table of instances. One instance, 'pes1201800046' (i-0c8ea3520cf3bfa4e), is selected and shown in detail. The 'Storage' tab is active under the instance details, showing two volumes: '/dev/sda1' (root device) and '/dev/sdf' (block device). Both volumes are listed as 'Attached' in the 'Attachment status' column. The 'Volume ID' column lists 'vol-01850484b8802578' and 'vol-08000da15e30756d1' respectively. The 'Size' column shows 8 and 10 GB respectively. The 'Filesystem' column shows '/dev/sda1' and '/dev/sdf'. The 'Used' column shows 476M and 0 respectively. The 'Avail' column shows 476M and 98M respectively. The 'Use%' column shows 0% and 1% respectively. The 'Mounted on' column shows '/dev' and '/run' respectively. The 'Filesystem' column also lists '/dev/xvda1', '/dev/loop0', '/dev/loop1', and '/dev/xvdf' with their respective details.

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.

```
ubuntu@ip-172-31-58-237:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            476M   476M    0% /dev
tmpfs           98M   756K   98M  1% /run
/dev/xvda1      7.7G  1.2G  6.6G 15% /
tmpfs           490M   0     490M  0% /dev/shm
tmpfs           5.0M   0     5.0M  0% /run/lock
tmpfs           490M   0     490M  0% /sys/fs/cgroup
/dev/loop0       98M   98M   0  100% /snap/core/10185
/dev/loop1       29M   29M   0  100% /snap/amazon-ssm-agent/2012
tmpfs           98M   0     98M  0% /run/user/1000
/dev/xvdf      9.8G  37K  9.3G  1% /mnt/data-store
ubuntu@ip-172-31-58-237:~$
```

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

The screenshot shows the AWS EC2 Management Console with the 'Schemas' tab selected. On the left, the navigation menu includes 'Instances' under 'Elastic Block Store'. In the main content area, a table lists a single snapshot:

Name	Snapshot ID	Size	Description	Status	Started	Progress	Encryption
SNAPSHOT	snap-0ac0a610207...	10 GiB	Snapshot of my volume	completed	January 22, 2021 at 1:56:16 ...	available (100%)	Not Encrypted

Below the table, a detailed view of the snapshot 'snap-0ac0a610207055c7c' is shown, including its status, volume information, and description.

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

The screenshot shows the AWS EC2 Management Console with the 'Volumes' tab selected. On the left, the navigation menu includes 'Volumes' under 'Elastic Block Store'. In the main content area, a table lists volumes:

Name	Volume ID	Size	Volume Type	IOPS	Throughput	Snapshot	Created	Availability Zone	State	Alarm Status	Attachment Information
SNAPSHOT	vol-0688a43d...	10 GiB	gp2	100	-	snap-0ac0a61...	January 22, 2021 at ...	us-east-1e	available	None	i-0deaa3520cf5bfa4...
pes1201800...	vol-080000de...	8 GiB	gp2	100	-	snap-0b071e0...	January 22, 2021 at ...	us-east-1e	in-use	None	i-0deaa3520cf5bfa4...
		10 GiB	gp2	100	-		January 22, 2021 at ...	us-east-1e	in-use	None	i-0deaa3520cf5bfa4...

Below the table, a detailed view of the volume 'vol-0688a43d7adc0e71' is shown, including its status, snapshot, and attachment information.

Task D:

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

The screenshot shows a Microsoft Excel spreadsheet with the title bar "Fri Jan 22, 03:00 • new-report.png (1045x602) - Chromium". The spreadsheet contains a table with the following columns: Service, Operation, UsageType, Resource, StartTime, EndTime, and UsageValue. The data includes various AWS services like AmazonS3, PutObject, HeadBucket, and GetObjectForReplication, along with their respective metrics and timestamps.

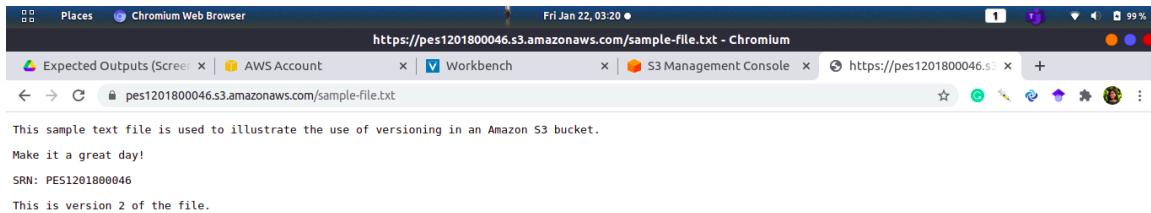
Service	Operation	UsageType	Resource	StartTime	EndTime	UsageValue
AmazonS3	HeadBucket	USW2-C3DataTransfer-Out-Bytes	lab-test-bucket-77	10/31/2020 0:00	12/31/2020 11:59	15309
AmazonS3	PutObject	USW2-C3DataTransfer-In-Bytes	admin-test-77	10/31/2020 0:00	12/31/2020 11:59	19032
AmazonS3	HeadBucket	USW2-Requests-Tier2	admin-test-77	10/31/2020 0:00	12/31/2020 11:59	128
AmazonS3	PutObjectForReplication	USW1-Requestst-SIA-Tier1	mybucket-98765	10/31/2020 0:00	12/31/2020 11:59	56888
AmazonS3	GetObjectFor Replication	USW1-USW2-AWS-In-Bytes	mybucket-98766	10/31/2020 0:00	12/31/2020 11:59	254587
AmazonS3	GetObjectFor Replication	USW2-C3DataTransfer-Out-Bytes	mybucket-98767	10/31/2020 0:00	12/31/2020 11:59	235
AmazonS3	HeadBucket	USW2-C3DataTransfer-In-Bytes	mybucket-98768	10/31/2020 0:00	12/31/2020 11:59	25589
AmazonS3	PutObject	USW2-Requests-Tier2	mybucket-98769	10/31/2020 0:00	12/31/2020 11:59	2348
AmazonS3	PutObjectForReplication	USW1-Requestst-SIA-Tier1	mybucket-98770	10/31/2020 0:00	12/31/2020 11:59	15309
AmazonS3	GetObjectForReplication	USW1-USW2-AWS-In-Bytes	mybucket-98771	10/31/2020 0:00	12/31/2020 11:59	19032
AmazonS3	GetObjectForReplication	USW2-C3DataTransfer-Out-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	128
AmazonS3	HeadBucket	USW2-C3DataTransfer-In-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	56888
AmazonS3	PutObject	USW2-Requests-Tier2	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	254587
AmazonS3	PutObjectForReplication	USW1-Requestst-SIA-Tier1	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	235
AmazonS3	GetObjectForReplication	USW1-USW2-AWS-In-Bytes	lab-example-bucket	10/31/2020 0:00	12/31/2020 11:59	25589

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

The screenshot shows the AWS S3 Management Console with the URL "https://pes1201800046.s3.console.aws.amazon.com/s3/buckets/pes1201800046?region=us-east-1&tab=objects&showversions=true". The page displays three versions of the file "sample-file.txt" with the following details:

Name	Type	Version ID	Last modified	Size	Storage class
sample-file.txt	txt	RxpdcchuTmAIXQoJ5lUvsB9rlTaMmT9LZ	January 22, 2021, 03:18:41 (UTC+05:30)	164.0 B	Standard
sample-file.txt	txt	Xf9RzdfPu6GkjcnMdJI_DfcxfMLgj4bZ	January 22, 2021, 03:16:47 (UTC+05:30)	132.0 B	Standard
new-report.png	png	null	January 22, 2021, 02:57:14 (UTC+05:30)	84.0 KB	Standard

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 overcommitting 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 Purpose SSD EBS Volumes 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 Availability 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 volume's lifetime be bound to the lifetime of 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.

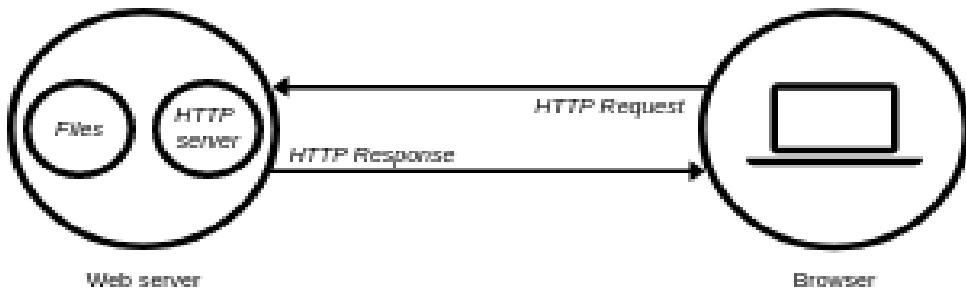
Sub Task 1: Launch a AWS EC2 Instance

- 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:

- [Apachebenchmark](#)
- [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:

```
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": "qwert",
    "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

The screenshot shows the AWS EC2 Management Console interface. The main content area displays the 'Instance summary' for an instance with the ID i-0c31def6e6862743a. The instance is listed as 'Running'. Key details shown include its Public IPv4 address (3.237.24.142), Public IPv4 DNS (ec2-3-237-24-142.compute-1.amazonaws.com), and Private IPv4 address (172.31.72.166). The instance is associated with a VPC (vpc-18983f65) and a subnet (subnet-39eeaa337). The IAM role is listed as 'arn:aws:sts::309504971180:assumed-role/vocstartsoft/user1181272+ruchkashashidhara@pesu.pes.edu'. The instance type is t2.medium. The status bar at the bottom indicates the browser is running on a Mac with a battery level of 63%.

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

2a.png showing that apache server status is active

```

File Edit View Search Terminal Help
ubuntu@ip-172-31-72-166:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Drop-In: /lib/systemd/system/apache2.service.d
             └─apache2-systemd.conf
     Active: active (running) since Wed 2021-01-27 16:00:16 UTC; 1min 23s ago
       Main PID: 18472 (apache2)
      Tasks: 55 (limit: 4686)
     CGroup: /system.slice/apache2.service
             ├─18472 /usr/sbin/apache2 -k start
             ├─18474 /usr/sbin/apache2 -k start
             ├─18475 /usr/sbin/apache2 -k start
             └─18476 /usr/sbin/apache2 -k start

Jan 27 16:00:16 ip-172-31-72-166 systemd[1]: Starting The Apache HTTP Server...
Jan 27 16:00:16 ip-172-31-72-166 systemd[1]: Started The Apache HTTP Server.
ubuntu@ip-172-31-72-166:~$ 

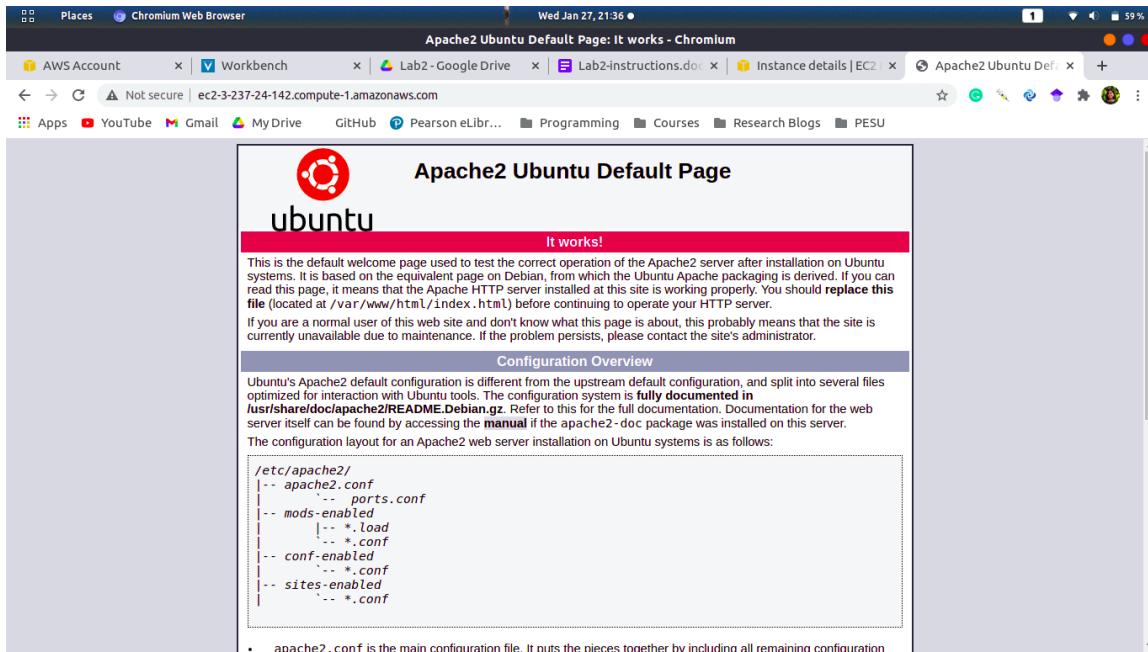
```

Sub Task 3: Setup Firewall Rules

3a.png showing the list of rules updated

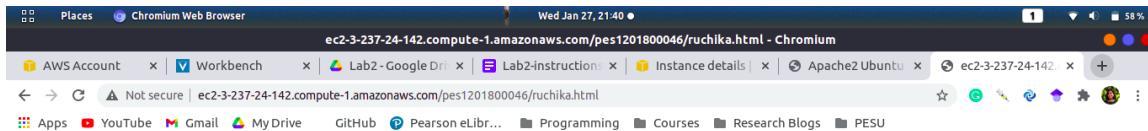
Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-

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

```
Places Terminal Wed Jan 27, 22:13 ● 52 %
File Edit View Search Terminal Tabs Help asys@asys-ruchika: ~
ubuntu@ip-172-31-72-166:/var/www/html/pes1201800046 x asys@asys-ruchika: ~
asys@asys-ruchika:~$ ab -k -c 40 -n 10000 http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking ec2-3-237-24-142.compute-1.amazonaws.com (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Apache/2.4.29
Server Hostname:     ec2-3-237-24-142.compute-1.amazonaws.com
Server Port:          80

Document Path:        /pes1201800046/ruchika.html
Document Length:      106 bytes

Concurrency Level:   40
Time taken for tests: 56.800 seconds
Complete requests:   10000
Failed requests:    0
Keep-Alive requests: 9920
Total transferred:  4116520 bytes
HTML transferred:   1060000 bytes
Requests per second: 176.06 [#/sec] (mean)
Time per request:   227.199 [ms] (mean)
Time per request:   5.680 [ms] (mean, across all concurrent requests)
```

```
Places Terminal Wed Jan 27, 22:13 ● 52 %
File Edit View Search Terminal Tabs Help asys@asys-ruchika: ~
ubuntu@ip-172-31-72-166:/var/www/html/pes1201800046 x asys@asys-ruchika: ~
asys@asys-ruchika:~$ ab -k -c 40 -n 10000 http://ec2-3-237-24-142.compute-1.amazonaws.com/pes1201800046/ruchika.html
Server Hostname:     ec2-3-237-24-142.compute-1.amazonaws.com
Server Port:          80

Document Path:        /pes1201800046/ruchika.html
Document Length:      106 bytes

Concurrency Level:   40
Time taken for tests: 56.800 seconds
Complete requests:   10000
Failed requests:    0
Keep-Alive requests: 9920
Total transferred:  4116520 bytes
HTML transferred:   1060000 bytes
Requests per second: 176.06 [#/sec] (mean)
Time per request:   227.199 [ms] (mean)
Time per request:   5.680 [ms] (mean, across all concurrent requests)
Transfer rate:       70.78 [kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3  24.5     0   257
Processing:   210   224  8.6    223   310
Waiting:      210   224  8.6    223   310
Total:        210   227  26.0    223   489

Percentage of the requests served within a certain time (ms)
  50%   223
  66%   226
  75%   228
  80%   230
  90%   233
  95%   240
  98%   259
  99%   434
100%   489 (longest request)
asys@asys-ruchika:~$
```

5b.png screenshot of the all set of graphs

Places Chromium Web Browser

Instances | EC2 Management Console - Chromium

console.aws.amazon.com/ec2/v2/home?region=us-east-1#instancesinstanceState=running

Search for services, features, marketplace products, and docs [Alt+S]

Apps YouTube Gmail My Drive GitHub Pearson eLibr... Programming Courses Research Blogs PESU

New EC2 Experience

EC2 Dashboard Events Tags Limits Instances Instances Now Instance Types Launch Templates Spot Requests Savings Plans Reserved Instances Dedicated Hosts Scheduled Instances Capacity Reservations Images AMIs Elastic Block Store Volumes Snapshots Lifecycle Manager

Instances (1/1) Instance state: running Clear filters

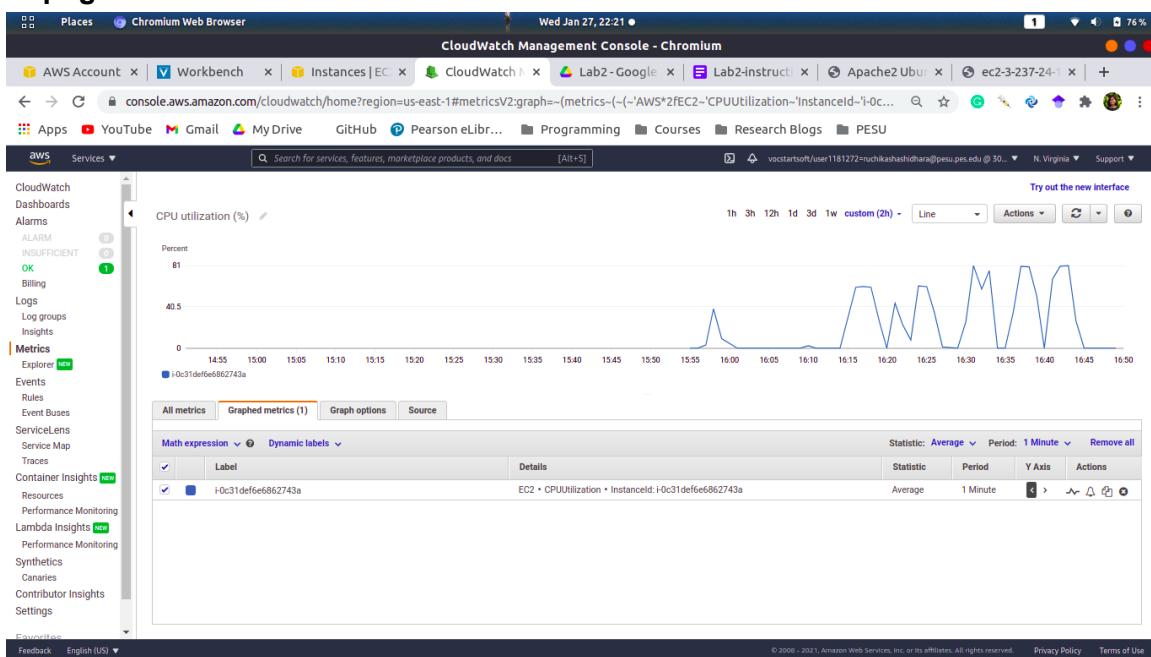
Name Instance ID Instance state Instance type Status check Alarm status Availability Zone Public IPv4 DNS Public IPv4 ...

- i-0c31def6e6862745a Running t2.medium 2/2 checks ... 1/2 h...+ us-east-1f ec2-5-257-24-142.co... 3.237.24.142

CPU utilization (%) Status check failed (any) (count) Status check failed (Instance) (count) Status check failed (system) (count)

Network in (bytes) Network out (bytes) Network packets in (count) Network packets out (count)

5c.png screenshot of CPU utilization



Task B:

Objective: Deploy Flask App on AWS Elastic Beanstalk

B1.png screenshot of Environment Health

The screenshot shows the AWS Elastic Beanstalk environment dashboard for 'Pes1201800046-env'. The left sidebar shows the environment configuration. The main panel displays the environment's health status as 'Green' with a checkmark icon. It shows the running version as 'Sample Application' and the platform as 'Python 3.6 running on 64bit Amazon Linux/2.9.18'. Below this, a table lists recent events, all of which are INFO level messages related to the environment's launch and configuration.

Time	Type	Details
2021-01-29 17:59:56 UTC+0530	INFO	Successfully launched environment: Pes1201800046-env
2021-01-29 17:59:56 UTC+0530	INFO	Application available at Pes1201800046-env.eba-cm2wpyve.us-east-1.elasticbeanstalk.com.
2021-01-29 17:59:51 UTC+0530	INFO	Added EC2 instance 'i-0f49250f4d9660a4d' to Auto Scaling Group 'awsseb-e-mxfusmmtc-stack-AWSEBAutoScalingGroup-1VOG59JSYBXMQ'.
2021-01-29 17:59:51 UTC+0530	INFO	Environment health has been set to GREEN
2021-01-29 17:59:51 UTC+0530	INFO	Adding instance 'i-0f49250f4d9660a4d' to your environment.

B2.png screenshot of DB configuration (RDS)

The screenshot shows the AWS RDS database configuration for 'aa1kvxsoyp67z8i'. The left sidebar shows the RDS service navigation. The main panel displays the database summary, showing the DB identifier 'aa1kvxsoyp67z8i', CPU usage of 2.30%, and other details like engine (MySQL Community) and region (us-east-1d). Below this, the 'Connectivity & security' tab is selected, showing the endpoint, networking (VPC and subnet group), and security (VPC security groups and certificate authority).

B3.png Register User Request (Postman screenshot)

The screenshot shows the Postman application interface. At the top, it displays the date and time: Fri Jan 29, 18:44. The main workspace shows an 'Untitled Request' for a POST method to the URL <http://pes1201800046-env.eba-cm2wpyve.us-east-1.elasticbeanstalk.com/api/register>. The 'Body' tab is selected, showing a JSON payload:

```
1 [{}]
2   "uname" : "pes1201800046",
3   "pswd" : "pes1201800046",
4   "age" : 20,
5   "mob" : "123456789"
6 ]
```

Below the body, the response status is 200 OK, time is 1288 ms, and size is 199 B. The response body is shown as an empty object: {}.

B4.png Verify User Request (Postman screenshot)

This screenshot of Postman shows a similar setup to the previous one, but for a different endpoint. It's an 'Untitled Request' for a POST method to the URL <http://pes1201800046-env.eba-cm2wpyve.us-east-1.elasticbeanstalk.com/api/verify>. The 'Body' tab is selected, showing a JSON payload:

```
1 [{}]
2   "uname" : "pes1201800046",
3   "pswd" : "pes1201800046"
4 ]
```

The response status is 200 OK, time is 279 ms, and size is 242 B. The response body is a JSON object containing user information: {"id": 1, "uname": "pes1201800046", "pswd": "pes1201800046", "age": 20, "mob": "123456789"}.

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

Reading – 20 mins

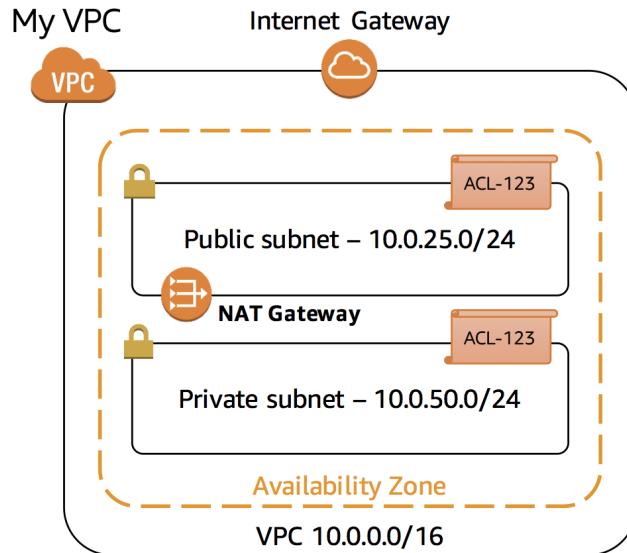
- [Virtual Private Cloud](#) : Make sure to go through benefits and use cases
- [What is Amazon VPC?](#) : Understand Key concepts for VPCs

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:
mins

-40

Migrating your existing application on Beanstalk into the VPC

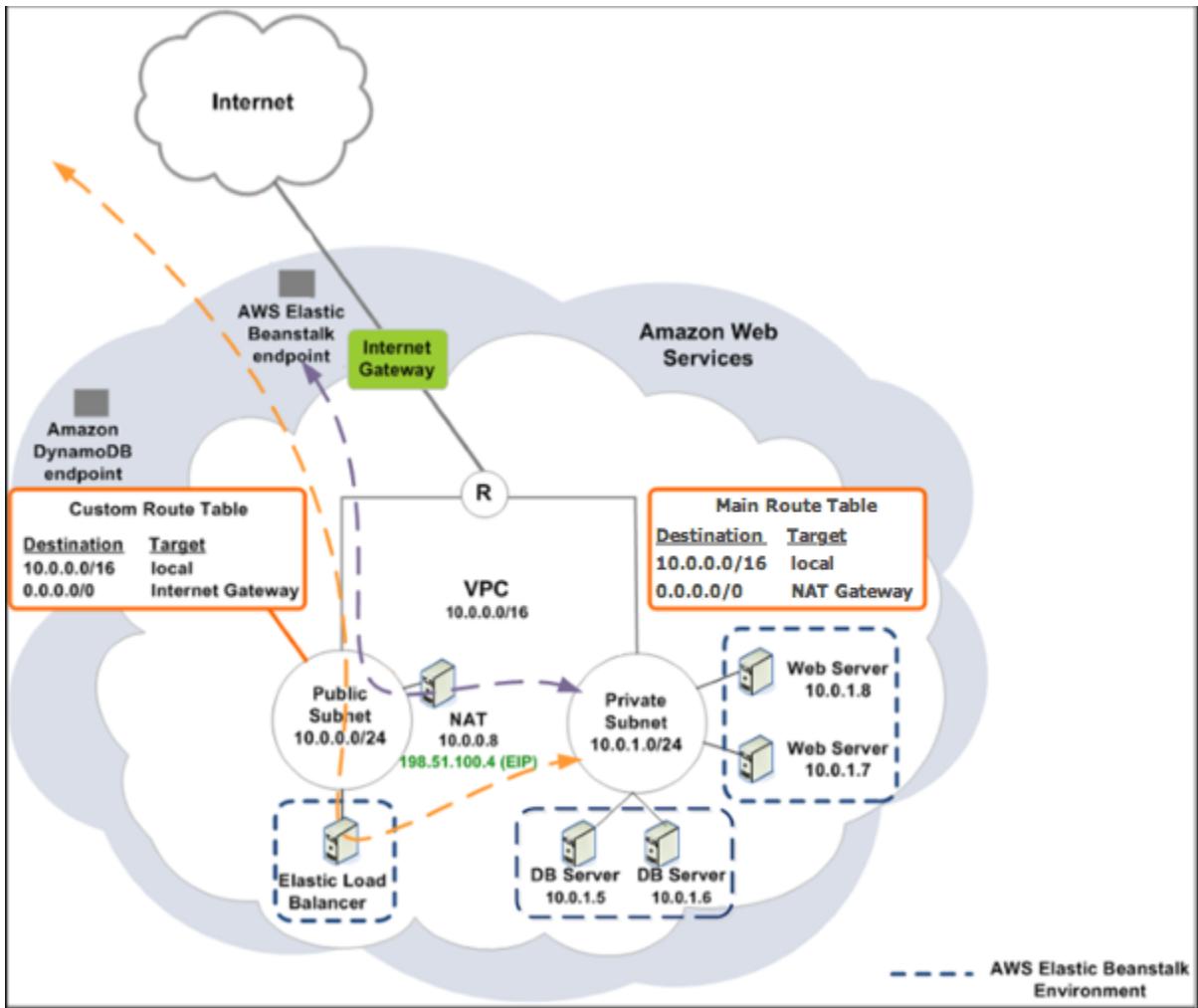
(Note: Refer the pictorial representation below)

- 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
mins

-10

- 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

The screenshot shows the AWS Elastic IP Addresses page. A green banner at the top indicates "Elastic IP address allocated successfully." Below the banner, the table lists one item: a Public IP address (52.207.151.99) associated with an allocation ID (eipalloc-016c30d218676f377). The details panel below shows the following information:

Allocated IPv4 address	52.207.151.99	Type	Public IP	Allocation ID	eipalloc-016c30d218676f377	Associated Instance ID	-	Private IP address	-	Association ID	-
Scope	VPC	Associated instance ID	-	Private IP address	-	Network interface ID	-	NAT Gateway ID	-	Address pool	Amazon
Network interface owner account ID	-	Public DNS	-	Allocation ID	eipalloc-016c30d218676f377	Association ID	-	Network interface ID	-	Address pool	Amazon
Network Border Group	us-east-1										

Task 2: Create VPC

3b.png - VPC Details

The screenshot shows the AWS Your VPCs page. It displays two VPCs: "RUCHIKA_VPC" (VPC ID: vpc-053286c2d8762641f) and another unnamed VPC (VPC ID: vpc-18983f65). The "RUCHIKA_VPC" row is selected, showing its details in the expanded panel below:

VPC ID	vpc-053286c2d8762641f	State	Available	IPv4 CIDR	10.0.0.0/16	IPv6 CIDR (Network border group)	-	IPv6 pool	-
Tenancy	Default	DHCP options set	dopt-1044046a	Main route table	rtb-0b173e59f116a94ac			Main network ACL	acl-0e76be4f3d66b3de8
Default VPC	No	IPv4 CIDR	10.0.0.0/16	IPv6 pool	-			IPv6 CIDR (Network border group)	-
Owner ID	309504971180								

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

The screenshot shows the AWS VPC Subnets list. There are two subnets listed:

Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR
Public Subnet	subnet-0fa4971b00d9a43c5	Available	vpc-055286c2d8762641f RUCHIKA_VPC	10.0.0.0/24	-
Private Subnet	subnet-006d85a8c67b2b9a6	Available	vpc-18983f65	172.31.0.0/20	-

3d.png - Public Network Details

The screenshot shows the AWS VPC Subnet details page for the subnet-006d85a8c67b2b9a6 Public Subnet.

Details

Subnet ID subnet-006d85a8c67b2b9a6	State Available	VPC vpc-055286c2d8762641f RUCHIKA_VPC	IPv4 CIDR 10.0.0.0/24
Available IPv4 addresses 250	IPv6 CIDR -	Availability Zone us-east-1a	Availability Zone ID use1-az4
Network border group us-east-1	Route table rtb-078ae81ab0979ef3f	Network ACL acl-0e76be4f3d6b5de8	Default subnet No
Auto-assign public IPv4 address No	Auto-assign IPv6 address No	Auto-assign customer-owned IPv4 address No	Customer-owned IPv4 pool -
Outpost ID -	Owner 309504971180	Subnet ARN arn:aws:ec2:us-east-1:309504971180:subnet-006d85a8c67b2b9a6	

Actions

Flow logs | Route table | Network ACL | Tags | Sharing

Task 3: Analyse the VPC

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

The screenshot shows the AWS VPC Network ACL details page for 'acl-0e76be4f3d66b3de8'. The 'Details' tab is selected, displaying information such as Network ACL ID, Associated with 2 Subnets, Default Yes, and VPC ID vpc-053286c2d8762641f / RUCHIKA_VPC. The 'Owner' is listed as 309504971180. Below this, the 'Inbound rules' tab is selected, showing two rules:

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	HTTP (80)	TCP (6)	80	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

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

The screenshot shows the AWS Elastic Beanstalk environment configuration page for 'Pes1201800046app-env'. The left sidebar shows options like Go to environment, Configuration (selected), Logs, Health, Monitoring, Alarms, Managed updates, Events, and Tags. The main content area displays environment settings:

Managed updates	Managed updates: enabled Update level: Minor and patch Weekly update window: Sat:22:00
Notifications	Email: --
Network	Database subnets: subnet-0fa4971b00d9a43c3 Instance subnets: subnet-0fa4971b00d9a43c3 Public IP address: enabled VPC: vpc-053286c2d8762641f Visibility: public
Database	Endpoint: aanvgwnuppdumv Availability: Low (one AZ) Engine: mysql Instance class: db.t2.micro Retention: Create snapshot Storage: 5 Username: pes1201800046

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 [yourusername]-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

The screenshot shows the AWS SNS Subscriptions page with the following details:

ID	Endpoint	Status	Protocol	Topic
6984970d-6c89-4fdf-9e9c-fc8db5a32cc6	ms.ruchika23@gmail.com	Confirmed	EMAIL	PES1201800046-RUCHIKA
5da128eb-72ff-495f-8187-d980366416bf	ruchikashashidhara@pesu.pes.edu	Confirmed	EMAIL	PES1201800046-RUCHIKA
c675f8f1-1b67-4aa7-9791-7db931c647cc	https://event.vocareum.com/cloudwatch/calarms.php	Confirmed	HTTPS	vocAlarmTopic

4b.png - SNS Messages - Mail Received

The screenshot shows an email from AWS Notifications to the user. The email content is as follows:

Hello, this is Ruchika from SNS!

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:309504971180:PES1201800046-RUCHIKA:6984970d-6c89-4fdf-9e9c-fc8db5a32cc6&Endpoint=ms_ruchika23@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

The screenshot shows a second email from AWS Notifications to the user. The email content is as follows:

Hello, this is Ruchika from SNS!

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:309504971180:PES1201800046-RUCHIKA:5da128eb-72ff-495f-8187-d980366416bf&Endpoint=ruchikashashidhara@pesu.pes.edu>

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Task 2: Play around with SQS Queries

4c.png - SQS Standard Queue

The screenshot shows the AWS SQS Standard Queue 'RUCHIKA-QUEUE' details page. At the top, a green banner says 'Queue RUCHIKA-QUEUE created successfully. You can now send and receive messages.' Below the banner, the queue name 'RUCHIKA-QUEUE' is displayed. A navigation bar includes 'Edit', 'Delete', 'Purge', and 'Send and receive messages' buttons. The 'Details' tab is selected, showing the following configuration:

Name	Type	ARN
RUCHIKA-QUEUE	Standard	arn:aws:sqs:us-east-1:309504971180:RUCHIKA-QUEUE
Encryption	URL	Dead-letter queue
-	https://sns.us-east-1.amazonaws.com/309504971180/RUCHIKA-QUEUE	-

At the bottom, there are links for 'Feedback', 'English (US)', and legal notices.

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

The screenshot shows the 'Receive messages' page for the 'RUCHIKA-QUEUE'. The 'Messages available' section shows 4 messages. The 'Polling duration' is set to 30 seconds, and the 'Maximum message count' is 10. The 'Polling progress' is at 4 receives/second. Below this, a table lists 4 messages:

ID	Sent	Size	Receive count
e111a360-226e-473f-9377-4967c8682a95	12/02/2021, 21:31:00 GMT+5:30	17 bytes	4
c816c983-8b89-4e0d-870b-a344493995d8	12/02/2021, 21:31:45 GMT+5:30	17 bytes	2
69cef39e-089e-427d-a94e-b6ec3a8675fb	12/02/2021, 21:32:05 GMT+5:30	17 bytes	2
5e08c82b-01d4-42c1-95ff-b665c6bd74f0	12/02/2021, 21:32:15 GMT+5:30	17 bytes	2

At the bottom, there are links for 'Feedback', 'English (US)', and legal notices.

Task 3: Using Amazon SNS & Amazon SQS together

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

The screenshot shows the AWS Lambda function configuration interface. In the center, there is a modal window titled "Message: 2a83c848-8bd5-406e-afa3-91a25a35e650". The "Body" tab is selected, displaying a large JSON object. The JSON content includes fields like "Type", "MessageId", "TopicArn", "Subject", "Message", "Timestamp", "SignatureVersion", and "Signature". The "Signature" field contains a long string of characters. At the bottom right of the modal is a "Done" button. To the left of the modal, there is a sidebar with tabs for "Receive messages" and "Messages (5)". The "Messages (5)" tab is active, showing a list of five messages with their IDs: e111a360-226e-4, 4567c0682a95, 2a83c040-8bd5-4, 91a25a35e650, and cd16c983-8b89-4. On the far left, there is a "Feedback" section and language selection ("English (US)"). At the bottom of the interface, there are links for "Privacy Policy" and "Terms of Use".

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:

```
#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 from docker:

```
docker pull ubuntu:18.04
docker pull nginx:latest
docker pull python:latest
docker pull mongo:latest
```

Inside a folder task2 run 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("Fetched 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 mongodb 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("Fetched 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-compose 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

```
ubuntu@ip-172-31-65-205:~$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
 executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
 to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
ubuntu@ip-172-31-65-205:~$
```

Task 2: Docker images and Dockerfiles

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

```
ubuntu@ip-172-31-65-205:~/task2$ docker run task2
Running this inside a container!
My SRN is PES1201800046
ubuntu@ip-172-31-65-205:~/task2$
```

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



My SRN is PES1201800046

I am running a nginx container!

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

```
ubuntu@ip-172-31-65-205:~/task3a$ docker run -p 80:80 pes1201800046
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
49.206.2.216 - - [18/Feb/2021:19:29:28 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36"
49.206.2.216 - - [18/Feb/2021:19:29:35 +0000] "GET /my.html HTTP/1.1" 200 110 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36"
```

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

```
ubuntu@ip-172-31-65-205:~/task3$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f2c00252a64e mongo:latest "docker-entrypoint.s..." 28 minutes ago Up 28 minutes 0.0.0.0:27017->27017/tcp vigilant_yallow
ubuntu@ip-172-31-65-205:~/task3$ docker run task3
Inserted into the MongoDB database!
Fetched from MongoDB: {'_id': ObjectId('602ebc18d3cf208fc9e6ab7d'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
ubuntu@ip-172-31-65-205:~/task3$
```

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

```
ubuntu@ip-172-31-65-205:~/task3$ docker run -dp 27017:27017 --name mongodb --network=my-bridge-network mongo:latest
f7d7e52f120f6298f4381f8c5975028a93f309d68df84624f67cabbe24222
ubuntu@ip-172-31-65-205:~/task3$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f7d7e52f120f mongo:latest "docker-entrypoint.s..." 5 seconds ago Up 4 seconds 0.0.0.0:27017->27017/tcp mongodb
ubuntu@ip-172-31-65-205:~/task3$ ls
```

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

```
ubuntu@ip-172-31-65-205:~/task3$ docker run --network=my-bridge-network task3
Inserted into the MongoDB database!
Fetched from MongoDB: {'_id': ObjectId('602ec5d8639cf08ecc69ab61'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
ubuntu@ip-172-31-65-205:~/task3$
```

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-compose build

docker-compose up

```
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.143+00:00"}, "s": "I", "c": "STORAGE", "id": 20320, "ctx": "conn2", "msg": "createCollection", "attr": {"remote": "172.19.0.3:46096", "namespace": "sample_db.sample_collection", "uidDisposition": "generated", "uuid": {"$oid": "598ba5dc0-f0f5-46eb-9998-1c16993a8cc6"}, "options": {}}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.144+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn3", "msg": "client metadata", "attr": {"remote": "172.19.0.3:46096", "client": "conn3", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.156+00:00"}, "s": "I", "c": "INDEX", "id": 20345, "ctx": "conn2", "msg": "Index build: done building", "attr": {"buildUUID": null, "namespace": "sample_db.sample_collection", "index": {"id": 0, "i": 0}}}]
pycode_1 | Inserted into the MongoDB database!
pycode_1 | Fetched from MongoDB: {_id: ObjectId('602eefb2b8276b20719b5f50'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.642+00:00"}, "s": "I", "c": "-", "id": 20883, "ctx": "conn1", "msg": "interrupted operation as its client disconnected", "attr": {"opId": 23}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.643+00:00"}, "s": "I", "c": "NETWORK", "id": 22944, "ctx": "conn1", "msg": "Connection ended", "attr": {"remote": "172.19.0.3:46092", "connectionId": 1, "connectionCount": 2}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.648+00:00"}, "s": "I", "c": "NETWORK", "id": 22944, "ctx": "conn2", "msg": "Connection ended", "attr": {"remote": "172.19.0.3:46094", "connectionId": 2, "connectionCount": 1}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:52:35.649+00:00"}, "s": "I", "c": "NETWORK", "id": 22944, "ctx": "conn3", "msg": "Connection ended", "attr": {"remote": "172.19.0.3:46096", "connectionId": 3, "connectionCount": 0}}
task4_pycode_1 exited with code 0
```

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

```
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.901+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn3", "msg": "client metadata", "attr": {"remote": "172.19.0.3:46106", "client": "conn3", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
pycode_1 | Inserted into the MongoDB database!
pycode_1 | Fetched from MongoDB: {_id: ObjectId('602eefb2b8276b20719b5f50'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.977+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.4:39194", "connectionId": 4, "connectionCount": 4}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.979+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn4", "msg": "client metadata", "attr": {"remote": "172.19.0.4:39194", "client": "conn4", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.980+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.5:39078", "connectionId": 5, "connectionCount": 5}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.980+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn5", "msg": "client metadata", "attr": {"remote": "172.19.0.5:39078", "client": "conn5", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.982+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.4:39198", "connectionId": 6, "connectionCount": 6}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.982+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.5:39082", "connectionId": 7, "connectionCount": 7}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.983+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn7", "msg": "client metadata", "attr": {"remote": "172.19.0.5:39082", "client": "conn7", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.983+00:00"}, "s": "I", "c": "NETWORK", "id": 51800, "ctx": "conn6", "msg": "client metadata", "attr": {"remote": "172.19.0.4:39198", "client": "conn6", "doc": {"driver": {"name": "PyMongo", "version": "3.11.3"}, "os": {"type": "Linux", "name": "Linux", "architecture": "x86_64"}, "version": "5.4.0-1037-aws"}, "platform": "CPython 3.9.1.final.0"}]}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.985+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.4:39202", "connectionId": 8, "connectionCount": 8}}
pycode_3 | Inserted into the MongoDB database!
pycode_3 | Fetched from MongoDB: {_id: ObjectId('602eefb2b8276b20719b5f50'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
pycode_2 | Inserted into the MongoDB database!
pycode_2 | Fetched from MongoDB: {_id: ObjectId('602eefb2b8276b20719b5f50'), 'Name': 'Ruchika', 'SRN': 'PES1201800046'}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.987+00:00"}, "s": "I", "c": "NETWORK", "id": 22944, "ctx": "conn8", "msg": "Connection ended", "attr": {"remote": "172.19.0.4:39202", "connectionId": 8, "connectionCount": 7}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.987+00:00"}, "s": "I", "c": "NETWORK", "id": 22943, "ctx": "listener", "msg": "Connection accepted", "attr": {"remote": "172.19.0.5:39086", "connectionId": 9, "connectionCount": 8}}
mongodb_1 | {"t": [{"$date": "2021-02-18T22:57:31.987+00:00"}, "s": "I", "c": "NETWORK", "id": 22944, "ctx": "conn9", "msg": "Connection ended", "attr": {"remote": "172.19.0.5:39086", "connectionId": 9, "connectionCount": 7}}
```

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](#).

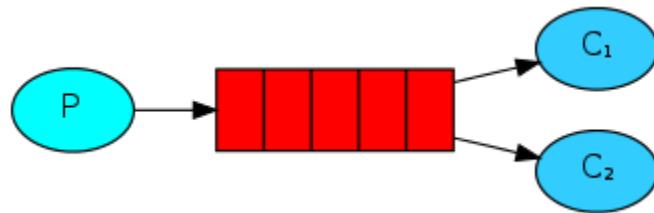
Prerequisites:

- 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.

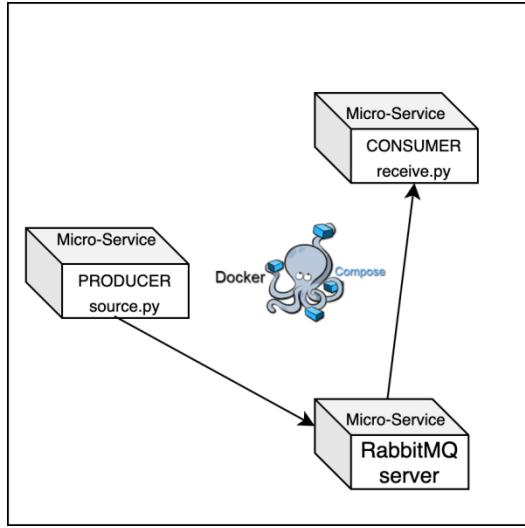


Background:

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 runs 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:

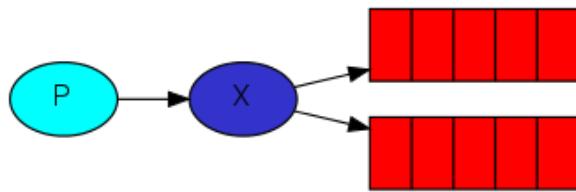
- please download the source folder given to you
- The source folder will have two directories namely “send” and “receive”. It will also consist of one empty yml file. Please maintain the directory structure given to you.
- The “send” and “receive” folder will have a Dockerfile.
- The “send” folder has “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

```
consumer_3 | [x] Done
consumer_4 | [x] Received 'Hello! My SRN is <PES1201800046> 0'
consumer_4 | [x] Done
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 8'
consumer_1 | [x] Received 'Hello! My SRN is <PES1201800046> 6'
consumer_1 | [x] Done
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 9'
consumer_1 | [x] Received 'Hello! My SRN is <PES1201800046> 10'
consumer_1 | [x] Done
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 10'
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 11'
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 12'
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 13'
producer_1 | [x] Sent 'Hello! My SRN is <PES1201800046> 14'
producer_2 | [x] Received 'Hello! My SRN is <PES1201800046> 3'
consumer_2 | [x] Done
consumer_2 | [x] Received 'Hello! My SRN is <PES1201800046> 7'
consumer_2 | [x] Done
consumer_2 | [x] Received 'Hello! My SRN is <PES1201800046> 11'
consumer_2 | [x] Done
consumer_2 | [x] Received 'Hello! My SRN is <PES1201800046> 5'
consumer_3 | [x] Done
consumer_3 | [x] Received 'Hello! My SRN is <PES1201800046> 9'
consumer_3 | [x] Done
consumer_4 | [x] Received 'Hello! My SRN is <PES1201800046> 4'
consumer_4 | [x] Done
consumer_4 | [x] Received 'Hello! My SRN is <PES1201800046> 8'
consumer_4 | [x] Done
consumer_4 | [x] Received 'Hello! My SRN is <PES1201800046> 12'
consumer_4 | [x] Done
consumer_3 | [x] Received 'Hello! My SRN is <PES1201800046> 13'
consumer_3 | [x] Done
consumer_1 | [x] Received 'Hello! My SRN is <PES1201800046> 14'
consumer_1 | [x] Done
rmq_1    | 2021-02-24 08:50:49.584 [info] <0.540.0> closing AMQP connection <0.540.0> (172.18.0.3:38594 -> 172.18.0.2:5672, vhost: '/', user: 'guest')
source_producer_1 exited with code 0
```

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

```
asya@PES1201800046-ruchika:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
9d97860591c1        source_consumer      "sh -c 'sleep 15 && ...'"   2 minutes ago     Exited (137) 18 seconds ago
6366db49b935        source_consumer      "sh -c 'sleep 15 && ...'"   2 minutes ago     Exited (137) 18 seconds ago
1889c0455137        source_consumer      "sh -c 'sleep 15 && ...'"   2 minutes ago     Exited (137) 18 seconds ago
1708d5960b45        source_producer      "sh -c 'sleep 20 && ...'"   2 minutes ago     Exited (0) About a minute ago
1349c32a6fc7        source_consumer      "sh -c 'sleep 15 && ...'"   2 minutes ago     Exited (137) 18 seconds ago
ff01977fef14        rabbitmq:3.8.3-alpine   "docker-entrypoint.s..."  10 minutes ago    Exited (0) 12 seconds ago
                                                              
source_consumer_4
source_consumer_3
source_consumer_1
source_producer_1
source_consumer_2
source_rmq_1
```

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 granted 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 granted 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 granted 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 granted 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 granted 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 granted 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: 'guest')
source_producer_1 exited with code 0
[]
```

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

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
d7ccb212e605	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
azd18a097884	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)

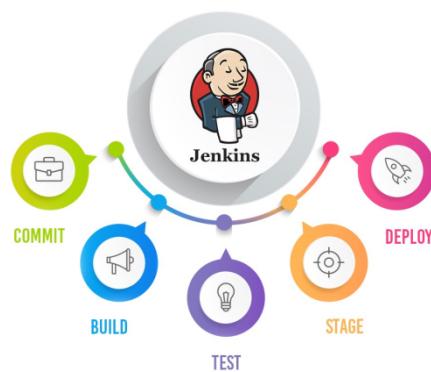
Introduction:

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.

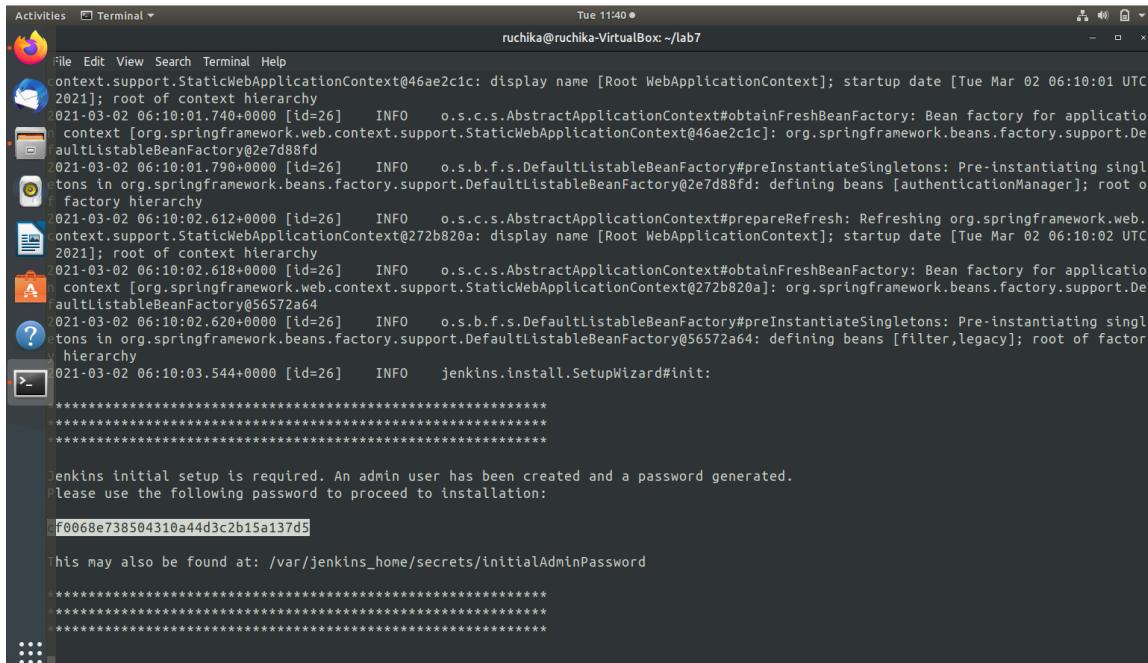
Steps:

- 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

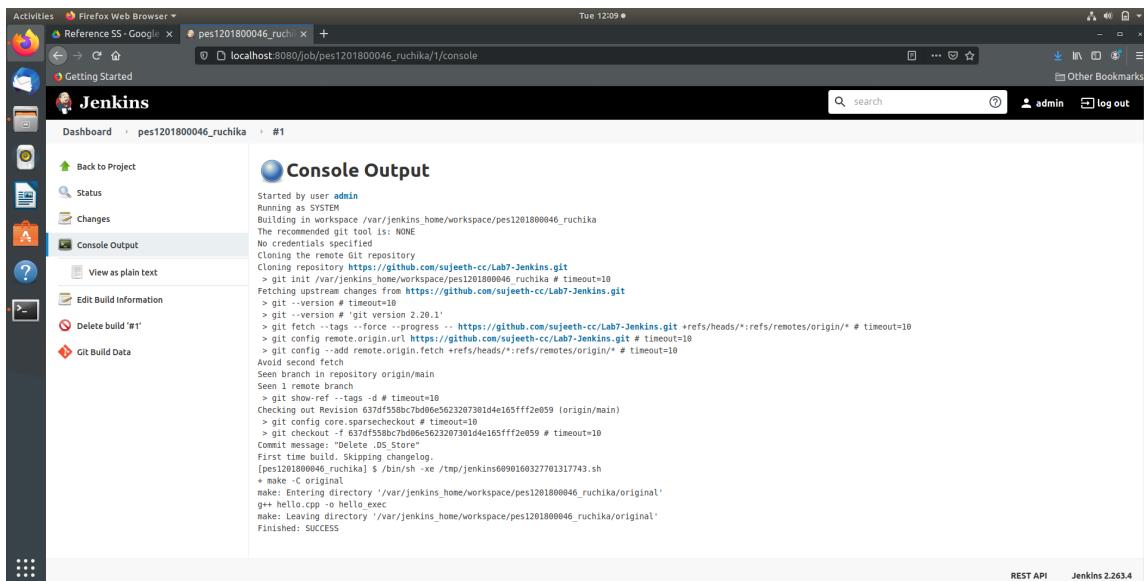
1A.png - SS of the running Docker Container



```
Tue 11:40 • ruchika@ruchika-VirtualBox: ~/lab
File Edit View Search Terminal Help
o.context.support.StaticWebApplicationContext@46ae2c1c: display name [Root WebApplicationContext]; startup date [Tue Mar 02 06:10:01 UTC 2021]; root of context hierarchy
2021-03-02 06:10:01.740+0000 [id=26] INFO o.s.c.s.AbstractApplicationContext#obtainFreshBeanFactory: Bean factory for application context [org.springframework.web.context.support.StaticWebApplicationContext@46ae2c1c]: org.springframework.beans.factory.support.DefaultListableBeanFactory@2e7d88fd
2021-03-02 06:10:01.790+0000 [id=26] INFO o.s.b.f.s.DefaultListableBeanFactory#preInstantiateSingletons: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@2e7d88fd: defining beans [authenticationManager]; root of factory hierarchy
2021-03-02 06:10:02.612+0000 [id=26] INFO o.s.c.s.AbstractApplicationContext#prepareRefresh: Refreshing org.springframework.web.context.support.StaticWebApplicationContext@272b820a: display name [Root WebApplicationContext]; startup date [Tue Mar 02 06:10:02 UTC 2021]; root of context hierarchy
2021-03-02 06:10:02.618+0000 [id=26] INFO o.s.c.s.AbstractApplicationContext#obtainFreshBeanFactory: Bean factory for application context [org.springframework.web.context.support.StaticWebApplicationContext@272b820a]: org.springframework.beans.factory.support.DefaultListableBeanFactory@56572a64
2021-03-02 06:10:02.620+0000 [id=26] INFO o.s.b.f.s.DefaultListableBeanFactory#preInstantiateSingletons: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@56572a64: defining beans [filter,legacy]; root of factory hierarchy
2021-03-02 06:10:03.544+0000 [id=26] INFO jenkins.install.SetupWizard#init:
*****
jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
f0068e738504310a44d3c2b15a137d5
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
```

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

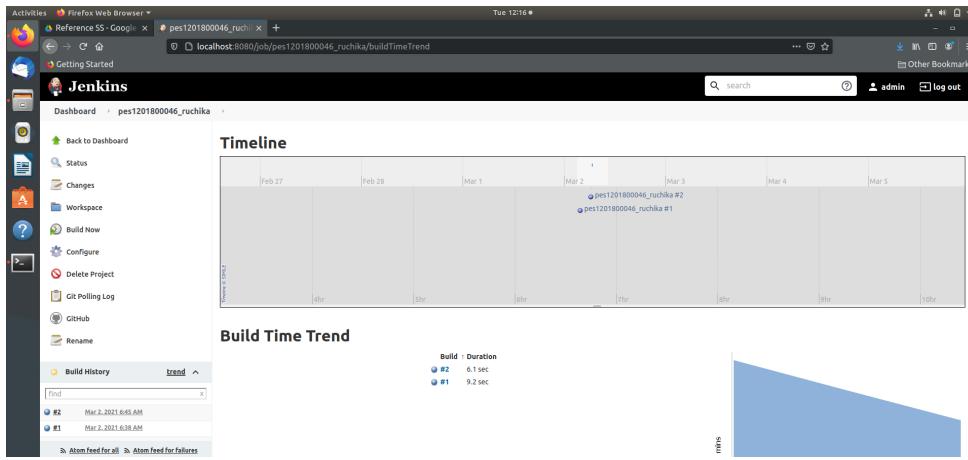
2A.png - SS of the console output showing build is successful of first job



The screenshot shows the Jenkins interface with a successful build log for a job named 'pes1201800046_ruchika'. The log output is as follows:

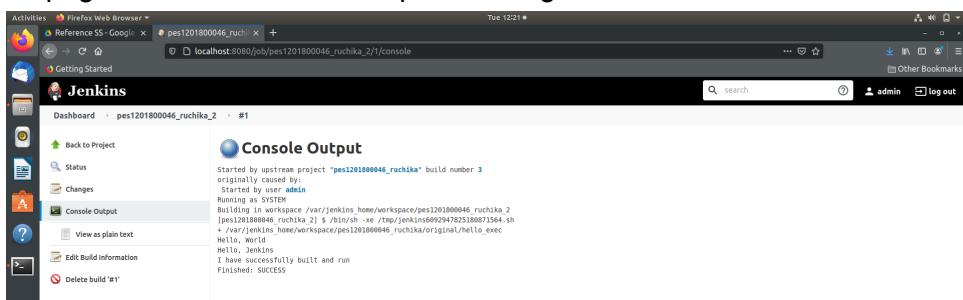
```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/pes1201800046_ruchika
The recommended git tool is: NONE
No credentials were provided
Configuring the remote Git repository
Closing repository https://github.com/sujeeth-cc/Lab7-Jenkins.git
> git init /var/jenkins_home/workspace/pes1201800046_ruchika # timeout=10
> git --version # timeout=10
> git --version # timeout=10
> git config --list --force --progress -- https://github.com/sujeeth-cc/Lab7-Jenkins.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/sujeeth-cc/Lab7-Jenkins.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
Seen branch main in repository origin/main
Seen 1 remote branch
Seen 1 remote branch
> git show-ref --tags -d # timeout=10
Counting objects: 6377, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (303/303), done.
Writing objects: 100% (6377/6377), done.
Total 6377 (delta 0), reused 0 (delta 0)
Checking connectivity... done.
> git config core.sparsecheckout # timeout=10
> git checkout -f 637df558bc7bd09e5623207301d4e105fff2e059 # timeout=10
Commit message: "Delete .DS_Store"
First time build. Skipping changelog.
[pes1201800046_ruchika] $ bin/sh -xe /tmp/jenkins5690160327701317743.sh
+ make
make: Entering directory '/var/jenkins_home/workspace/pes1201800046_ruchika/original'
g++ hello.cpp -o hello.exe
make: Leaving directory '/var/jenkins_home/workspace/pes1201800046_ruchika/original'
Finished: SUCCESS
```

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

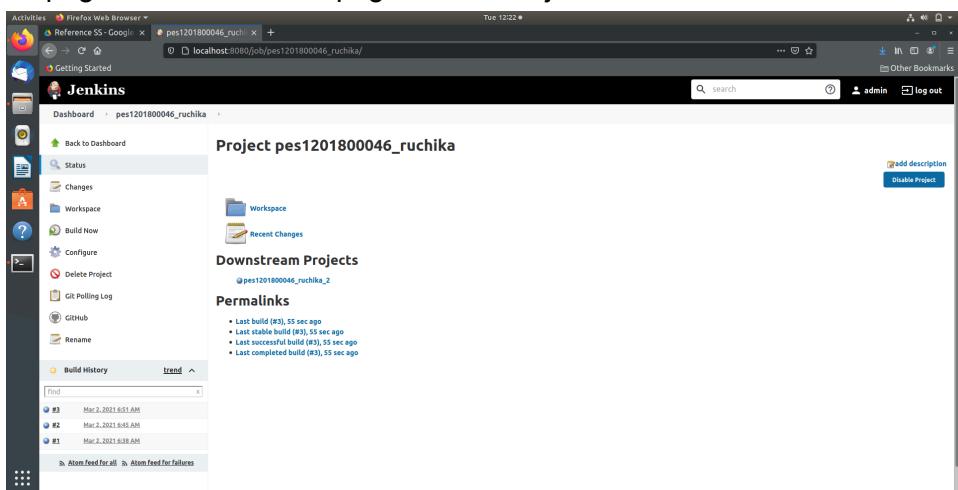


Task 3: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp

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

The screenshot shows the Jenkins dashboard with the 'Build History' section selected. The main area displays a timeline from February 27 to March 5, showing build activities. On March 2, four builds were triggered: 'pes1201800046_ruchika_2 #1', 'pes1201800046_ruchika #3', 'pes1201800046_ruchika #2', and 'pes1201800046_ruchika #1'. Below the timeline, a table titled 'Build Executor Status' lists four idle build executors.

Build	Time Since	Status
pes1201800046_ruchika_2 #1	1 min 42 sec	stable
pes1201800046_ruchika #3	1 min 51 sec	stable
pes1201800046_ruchika #2	7 min 41 sec	stable
pes1201800046_ruchika #1	14 min	stable

Legend: Atom feed for all Atom feed for failures Atom feed for just latest builds

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

The screenshot shows the Jenkins dashboard with the 'Build History' section selected. The main area displays a table titled 'All' showing two recent builds: 'pes1201800046_ruchika' and 'pes1201800046_ruchika_2'. Both builds were successful, with the first build having a duration of 4.1 seconds and the second having a duration of 84 ms.

S	W	Name	Last Success	Last Failure	Last Duration
		pes1201800046_ruchika	2 min 10 sec - #3	N/A	4.1 sec
		pes1201800046_ruchika_2	2 min 0 sec - #1	N/A	84 ms

Legend: Atom feed for all Atom feed for failures Atom feed for just latest builds

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 "$(cat 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, therefore a pod can contain 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
    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
      spec:
        containers:
          - name: mynginx
            image: nginx:latest
            resources:
              limits:
                memory: "128Mi"
                cpu: "500m"
            ports:
              - containerPort: 80
```

```

# This divider is a YAML specification to separate 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

The screenshot shows the AWS EC2 Management Console interface. At the top, there's a search bar with the query "Search for services, features, marketp [Alt+S]". Below the search bar, the main title is "Instances | EC2 Management Console - Chromium". The URL in the address bar is "console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances;search=pes1201800046". On the left, the AWS logo and "Services" dropdown are visible. The main content area displays a table titled "Instances (1/1) Info". The table has columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Z. One row is selected, showing "Name: pes1201800046", "Instance ID: i-0f98fd8aa2aaedef3a", "Instance state: Running", "Instance type: t2.medium", "Status check: 2/2 checks passed", "Alarm status: 1 alarms +", and "Availability Z: us-east-1b". Below the table, there are tabs for "Details", "Security", "Networking", "Storage", "Status checks", "Monitoring", and "Tags". Under the "Details" tab, there's a section titled "Instance summary" with two rows. The first row shows "Instance ID: i-0f98fd8aa2aaedef3a (pes1201800046)" and "Public IPv4 address: 54.91.247.52 | open address". The second row shows "Instance state: Running" and "Public IPv4 DNS: ec2-54-91-247-52.compute-1.amazonaws.com | open address". At the bottom of the page, there are links for "Feedback", "English (US)", "Privacy Policy", "Terms of Use", and "Cookie preferences".

Task 1: Create a Kubernetes cluster using minikube

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

```
ubuntu@ip-172-31-45-184:~$ minikube start
😄  minikube v1.18.1 on Ubuntu 18.04 (xen/amd64)
👍  Automatically selected the docker driver. Other choices: ssh, none
🌟  Starting control plane node minikube in cluster minikube
Pulling base image ...
⬇️  Downloading Kubernetes v1.20.2 preload ...
> preloaded-images-k8s-v9-v1....: 491.22 MiB / 491.22 MiB  100.00% 70.53 Mi
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
    ■ Generating certificates and keys ...
    ■ Booting up control plane ...
    ■ Configuring RBAC rules ...
💡  Verifying Kubernetes components...
    ■ Using image gcr.io/k8s-minikube/storage-provisioner:v4
💡  Enabled addons: storage-provisioner, default-storageclass
🎉  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
ubuntu@ip-172-31-45-184:~$ kubectl get node
NAME      STATUS    ROLES          AGE     VERSION
minikube  Ready     control-plane,master  67s    v1.20.2
ubuntu@ip-172-31-45-184:~$
```

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 with service

```
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 mins - 30

What is AWS EMR?

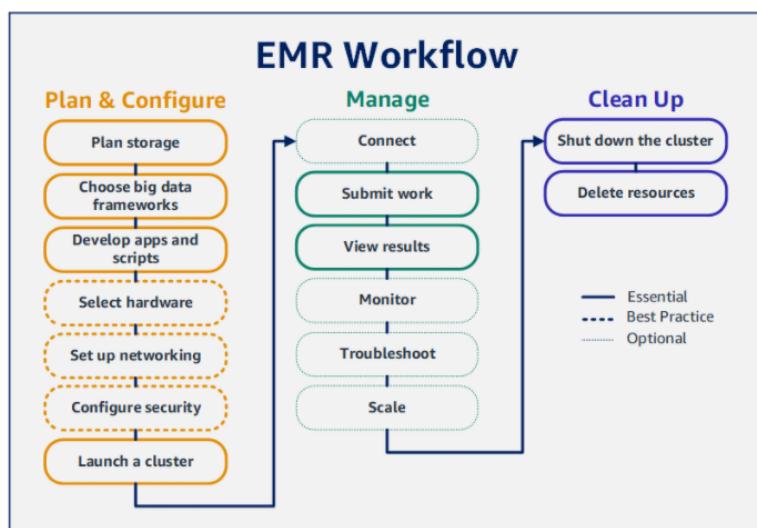
Amazon EMR is the industry-leading cloud big data platform for processing vast amounts of data using open source tools such as [Apache Spark](#), [Apache Hive](#), [Apache HBase](#), [Apache Flink](#), [Apache Hudi](#), and [Presto](#). Amazon EMR makes it easy to set up, operate, and scale your big 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 & healthViolations.py uploaded into the S3 Bucket

The screenshot shows the AWS S3 Management Console interface. At the top, there are three tabs: 'AWS Account', 'Workbench', and 'S3 Management Console'. The URL in the address bar is <https://s3.console.aws.amazon.com/s3/upload/pes1201800046-bucket?region=us-east-1>. A green banner at the top indicates 'Upload succeeded'.

Summary

Destination	Succeeded	Failed
s3://pes1201800046-bucket	2 files, 11.5 MB (100.00%)	0 files, 0 B (0%)

Files and folders (2 Total, 11.3 MB)

Name	Type	Size	Status	Error
food_establishment_data.csv	text/csv	11.3 MB	Succeeded	-
healthViolations.py	text/x-python	2.0 KB	Succeeded	-

9b.png - Screenshot of the running cluster

The screenshot shows the AWS EMR - AWS Console interface. The URL in the address bar is <https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#cluster-details:j-1Z8HSM81A7TSC>.

Cluster: my-cluster-pes1201800046 Waiting Cluster ready after last step completed.

Summary

ID: j-1Z8HSM81A7TSC	Release label: emr-6.2.0
Creation date: 2021-03-25 14:11 (UTC+5:30)	Hadoop distribution: Amazon
Elapsed time: 7 minutes	Applications: Spark 3.0.1, Zeppelin 0.9.0
After last step completes: Cluster waits	Log URI: s3://pes1201800046-bucket/logs/
Termination protection: Off Change	EMRFS consistent view: Disabled
Tags: -- View All / Edit	Custom AMI ID: --
Master public DNS: ec2-54-209-208-174.compute-1.amazonaws.com	Connect to the Master Node Using SSH

Application user interfaces

Persistent user interfaces:	Spark history server, YARN timeline server
On-cluster user interfaces:	Not Enabled Enable an SSH Connection

Network and hardware

Availability zone:	us-east-1a
Subnet ID:	subnet-0bb257d2146dc8751
Master:	Running 1 m5.xlarge
Core:	Running 2 m5.xlarge
Task:	--
Cluster scaling:	Not enabled

Security and access

Task 2: Managing Amazon EMR Cluster

9c.png - Screenshot of the Completed Spark Job

The screenshot shows the AWS EMR console interface. At the top, there are tabs for 'Places', 'AWS Account', 'Workbench', and 'EMR - AWS Console'. The URL in the address bar is <https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#cluster-details;j-1Z0...>. Below the tabs, there are buttons for 'Clone', 'Terminate', and 'AWS CLI export'. A message says 'Cluster: my-cluster-pes1201800046 Waiting Cluster ready after last step completed.' Below this, there are tabs for 'Summary', 'Application user interfaces', 'Monitoring', 'Hardware', 'Configurations', 'Events', 'Steps', and 'Bootstrap actions'. The 'Steps' tab is selected. It shows a table with one step:

ID	Name	Status	Start time (UTC+5:30)	Elapsed time	Log files
s-262TUMO183TDC	Spark-application-0046	Completed	2021-03-25 14:26 (UTC+5:30)	36 seconds	View logs

Details for the step:
JAR location : command-runner.jar
Main class : None
spark-submit --deploy-mode cluster s3://pes1201800046-bucket/health_violations.py --data_source s3://pes1201800046-bucket/food_establishment_data.csv --output_uri Arguments : s3://pes1201800046-bucket/myOutputFolder
Action on failure: Continue

At the bottom, there are buttons for 'Add step', 'Clone step', and 'Cancel step'. A link 'View Jobs in the Application History Tab' is also present.

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

The screenshot shows the AWS S3 Management Console. The URL in the address bar is <https://s3.console.aws.amazon.com/s3/buckets/pes1201800046-bucket?region=us-east-1&prefix=...>. The navigation bar shows 'Amazon S3 > pes1201800046-bucket > myOutputFolder/'. There is a 'Copy S3 URI' button. Below, there are tabs for 'Objects' and 'Properties', with 'Objects' selected.

The 'Objects (2)' section shows a table of objects:

Name	Type	Last modified	Size	Storage class
_SUCCESS	-	March 25, 2021, 14:26:41 (UTC+05:30)	0 B	Standard
part-00000-e9701a96-044c-4747-80a3-f15df268863-c000.csv	csv	March 25, 2021, 14:26:41 (UTC+05:30)	219.0 B	Standard

At the bottom, there are buttons for 'Feedback', 'English (US)', and links to 'Privacy Policy', 'Terms of Use', and 'Cookie preferences'.

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

The screenshot shows a LibreOffice Calc spreadsheet window titled "part-00000-e9701a96-044c-4747-80a3-f15d9f268863-c000.csv - LibreOffice Calc". The window has a dark theme. The data is displayed in a single sheet named "Sheet 1 of 1". The columns are labeled A through F, and the rows are numbered 1 through 14. Column A contains the restaurant names, and column B contains the total red violations. The data is as follows:

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

The screenshot shows the AWS S3 Management Console interface. On the left is a navigation sidebar with various icons. The main area is titled "Amazon S3". At the top, there are buttons for "Create bucket", "Copy ARN", "Empty", and "Delete". Below that is a search bar labeled "Find buckets by name". A table lists four buckets:

Name	AWS Region	Access	Creation date
Images-pes1201800046	US West (Oregon) us-west-2	Bucket and objects not public	March 30, 2021, 11:19:10 (UTC+05:30)
Images-pes1201800046-resized	US West (Oregon) us-west-2	Bucket and objects not public	March 30, 2021, 11:20:21 (UTC+05:30)
ql-cf-templates-1617083186-f9914ba976a9c9b0-us-west-2	US West (Oregon) us-west-2	Objects can be public	March 30, 2021, 11:16:27 (UTC+05:30)
qltrail-lab-2564-1617083199	US East (N. Virginia) us-east-1	Objects can be public	March 30, 2021, 11:16:41 (UTC+05:30)

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

The screenshot shows the AWS Lambda function configuration page for "Create-Thumbnail". The top navigation bar includes tabs for "Code", "Test", "Monitor", "Configuration", "Aliases", and "Versions". The "Code source" section is visible at the bottom. The "Designer" tab is active, showing a diagram with a green "S3" icon pointing to a blue "Create-Thumbnail" box. The "Create-Thumbnail" box has a "Layers" section with "(0)". To the right of the diagram, there is a "Description" field, a "Last modified" field (showing "5 minutes ago"), and a "Function ARN" field containing "arn:aws:lambda:us-west-2:738523773934:function:Create-Thumbnail".

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

The screenshot shows the AWS Lambda console interface. A modal window is open, displaying the execution results for a function named 'Create-Thumbnail'. The status is 'Execution result: succeeded (logs)'. The 'Details' section shows the returned value 'null'. The 'Summary' section provides performance metrics: Request ID (6e616c37-5039-4672-957f-ba7d3b190c68), Duration (939.16 ms), and Resources configured (128 MB). The 'Log output' section shows CloudWatch logs with entries for START, END, and REPORT requests, along with their respective durations and memory usage.

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

The screenshot shows the AWS S3 Management Console. The user is viewing the contents of a bucket named 'images-pes1201800046-resized'. The 'Objects' tab is selected, showing one object named 'HappyFace.jpg' which is a jpg file, last modified on March 30, 2021, at 11:38:18 (UTC+05:30), with a size of 2.6 KB and a storage class of Standard.

Name	Type	Last modified	Size	Storage class
HappyFace.jpg	jpg	March 30, 2021, 11:38:18 (UTC+05:30)	2.6 KB	Standard

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

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with various icons. The main area is titled 'FAQ' and displays a success message: 'The trigger FAQ-API was successfully added to function FAQ. The function is now receiving events from the trigger.' Below this, there's a 'Function overview' section with a thumbnail icon for 'FAQ', a 'Layers' section (empty), and a 'Triggers' section which lists one entry: 'API Gateway'. To the right of the triggers, there's a detailed view of the function: 'Description' (Provide a random FAQ), 'Last modified' (3 minutes ago), and 'Function ARN' (arn:aws:lambda:us-west-2:953432458474:function:FAQ). At the bottom, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration' (which is selected), 'Aliases', and 'Versions'. The 'Configuration' tab has sub-sections for 'General configuration' and 'Triggers (1)'. The 'Triggers' sub-section includes buttons for 'Enable', 'Disable', 'Fix errors', 'Delete', and 'Add trigger'. The status bar at the bottom indicates the browser is running Mozilla Firefox at 11:56 AM.

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

The screenshot shows a Mozilla Firefox browser window. The address bar contains the URL 'https://yr5do5xjc1.execute-api.us-west-2.amazonaws.com/m'. The main content area displays a JSON response: {"q": "What restrictions apply to AWS Lambda function code?", "a": "Lambda attempts to impose few restrictions on normal language and operating system activities, but there are a few activities that are disabled: Inbound network connections are managed by AWS Lambda, only TCP/IP sockets are supported, and ptrace (debugging) system calls are restricted. TCP port 25 traffic is also restricted as an anti-spam measure."}. Above the JSON output, there are tabs for 'JSON', 'Raw Data', and 'Headers', with 'JSON' being the active tab. The browser interface includes a toolbar with icons for back, forward, search, and other functions. The status bar at the bottom indicates the browser is running Mozilla Firefox at 11:58 AM.

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

The screenshot shows the AWS Lambda Function Overview page. The top navigation bar includes tabs for Introduction to Amazon, FAQ - Lambda, RuchikaS_PES12018000, and Screenshots - Google D. The main content area displays a function overview with tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Test tab is selected, showing an execution result with a success status. The log output section contains the following text:

```
Execution result: succeeded (logs)
Details
The area below shows the result returned by your function execution. Learn more about returning results from your function.

body: "(\\"q\\")\\nWhat languages does AWS Lambda support?\\n\\nAWS Lambda supports code written in Node.js (JavaScript), Python, and Java (Java 8 compatible). Your code can include existing libraries, even native ones. Lambda functions can easily launch processes using languages supported by Amazon Linux, including Bash, Go, and Ruby. Please read our documentation on using Node.js, Python and Java.\\n\\n"
Summary
Code SHA-256
8f6ac20Py7TnDvtUED1KZab0H1624YGS5CH+Gjnl=
Request ID
83895c00-ac81-40a9-9a39-0371157d6b7b
Duration
34.12 ms
Billed duration
35 ms
Resources configured
128 MB
Max memory used
65 MB
Log output
The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.
START RequestId: 83895c00-ac81-40a9-9a39-0371157d6b7b Version: $LATEST
2021-03-30T06:30:36,407Z          83895c00-ac81-40a9-9a39-0371157d6b7b  INFO  Quote selected: 4
2021-03-30T06:30:36,407Z          83895c00-ac81-40a9-9a39-0371157d6b7b  INFO  [
body: '(\\"q\\")\\nWhat languages does AWS Lambda support?\\n\\nAWS Lambda supports code written in Node.js (JavaScript), Python, and Java (Java 8 compatible). Your code can include existing libraries, even native ones. Lambda functions can easily launch processes using languages supported by Amazon Linux, including Bash, Go, and Ruby. Please read our documentation on using Node.js, Python and Java.\\n\\n"
]
END RequestId: 83895c00-ac81-40a9-9a39-0371157d6b7b
REPORT RequestId: 83895c00-ac81-40a9-9a39-0371157d6b7b Duration: 34.12 ms    Billed Duration: 35 ms   Memory Size: 128 MB   Max Memory Used: 65 MB
```

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://123456789012signin.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

The screenshot shows the AWS Identity and Access Management (IAM) Groups page. On the left, there's a navigation sidebar with options like Dashboard, Access management (Groups, Users, Roles, Policies), Access reports (Archive rules, Analyzers, Settings), Credential report, and Organization activity. The main content area has a search bar at the top. Below it, there are two buttons: 'Create New Group' and 'Group Actions'. A table titled 'Showing 4 results' lists four groups: EC2-Admin, EC2-Support, QLReadOnly, and S3-Support. Each group has 0 users, an inline policy, and was created on 2021-04-03 22:02 UTC+0530.

Group Name	Users	Inline Policy	Creation Time
EC2-Admin	0	✓	2021-04-03 22:02 UTC+0530
EC2-Support	0		2021-04-03 22:02 UTC+0530
QLReadOnly	1		2021-04-03 22:02 UTC+0530
S3-Support	0		2021-04-03 22:02 UTC+0530

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

This screenshot shows the 'Show Policy' dialog for the EC2-Support group. The dialog displays the JSON policy document. The policy allows various actions across different services like EC2, CloudWatch Metrics, and Auto Scaling. It includes statements for Describe operations on EC2, ELB, CloudWatch Metrics, CloudWatch Metrics Statistics, and Auto Scaling resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "elasticloadbalancing:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:Describe*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "autoscaling:Describe*",
      "Resource": "*"
    }
  ]
}
```

Task 2: Add Users to the Groups

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

The screenshot shows the AWS IAM service interface. On the left, the navigation pane is open, showing the 'Identity and Access Management (IAM)' section with 'Groups' selected under 'Access management'. The main content area displays a table titled 'Showing 5 results' with columns: 'User name', 'Groups', 'Access key age', 'Password age', 'Last activity', and 'MFA'. The data in the table is as follows:

User name	Groups	Access key age	Password age	Last activity	MFA
awsstudent	QLReadOnly	Today	Today	None	Not enabled
root-qwkl		None	None		
user-1	S3-Support	None	Today	None	Not enabled
user-2	EC2-Support	None	Today	None	Not enabled
user-3	EC2-Admin	None	Today	None	Not enabled

2b.png - Showing each group having one user

The screenshot shows the AWS IAM service interface. On the left, the navigation pane is open, showing the 'Identity and Access Management (IAM)' section with 'Groups' selected under 'Access management'. The main content area displays a table titled 'Showing 4 results' with columns: 'Group Name', 'Users', 'Inline Policy', and 'Creation Time'. The data in the table is as follows:

Group Name	Users	Inline Policy	Creation Time
EC2-Admin	1	✓	2021-04-03 22:02 UTC+0530
EC2-Support	1		2021-04-03 22:02 UTC+0530
QLReadOnly	1		2021-04-03 22:02 UTC+0530
S3-Support	1		2021-04-03 22:02 UTC+0530

Task 3: Testing User Access

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

The screenshot shows the AWS S3 console. The left sidebar is for 'Amazon S3' with options like Buckets, Access Points, Object Lambda Access Points, Batch Operations, and Access analyzer for S3. The main content area is titled 'qls-26745872-16a1b49d3929d85e-s3bucket-1cjrcic7zvfha'. The 'Properties' tab is active. In the 'Bucket overview' section, it shows the AWS Region as 'US West (Oregon) us-west-2', the ARN as 'arn:aws:s3:::qls-26745872-16a1b49d3929d85e-s3bucket-1cjrcic7zvfha', and the Creation date as 'April 3, 2021, 22:02:59 (UTC+05:30)'. The 'Bucket Versioning' section indicates 'Bucket Versioning Disabled'. At the bottom, there's a note about Multi-factor authentication (MFA) delete.

3b.png - Showing user-1 having NO access to view details of the EC2 Instances

The screenshot shows the AWS EC2 Instances page. The left sidebar has sections for 'New EC2 Experience', 'EC2 Dashboard', 'Events', 'Tags', 'Limits', 'Instances' (selected), 'Images', and 'Feedback'. The main content area is titled 'Instances' and shows a table with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone. A message at the top says 'You do not have any instances in this region'. Below the table, it says 'Select an instance above'.

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

The screenshot shows the AWS EC2 Instances page. A modal dialog box is open at the top left, displaying an error message: "Failed to stop the instance i-0d0cb5dd98173f60c". The message states: "You are not authorized to perform this operation. Encoded authorization failure message: VCJncmf9uxZj56ueuHQnjh8Duv-x3B-1QwKox7OsEGyOn7Xjd-X9j3wozOA6g998q69Fo3dEsaJdwvkkx0AeZO6xyN4_VBwiLkyLhzHTx6AwAxap5grnIu531UzCvM3qYmkSnrlP16wH0zjC0wuzYL5FeV9kIfheNIAcKh-A-brQ-2_AueO2zq3eP_dVOj3eD7GYY-n0m2sPv0pN3Acpq1CczjIC6X9qx_YUy13z8axI-2vPNxtza242xb7Dm-9Y71PLBSLEt4cfqszkobRcq3ewW5lqvF3pdnGOM17g6zCJSZ9bbAwFpZq4x98xZPeB0hcCs2dVnPnUzPxAZQRb14d0l8VGK8NkaZd4dLRoeQwxBoRgySwCkwcl7QwOBnjefAE6K14pPHUFEVP0fhQ5g5tUjkxKocoWH5434B6zDMG6KJ8Ku2H2SH5yOzBoux1_o5GdImSG_8isWDOnuWQHN7K8c9k_h7B9653cNcpXj6zFrhXX508Mo66GBGx5BbB9-dy21YQ6vE3y56psAeQ3y3t6RR1nJR8VbdutvtS2ysWJUL-cJ7fcFvnR8Gsg-p_5oQevR4WGX-9CbfNgQLBGYWWh7Cfy9AIOrAhhQ8VVV2uT1Kfc-59BXPlomv1cmjG6688S1fUXRwcSeDyazonePloYPC40cktMsOceGiqApOLMRgZ0o6m0eVvX8XjgNM4glFnHYSQKeXjwvhBoDpvOdJRdJ1xBcs8xjhshRRoftTpITMBAgeLpjQ1rT7fzIMPC2dd4Jxp5-35NhWKuLtZGferAlGMBQCirrgg1Dkm4r0nzoobwN_s1Y4Z2dZafI".

Instances (1/1)	Info
<input checked="" type="checkbox"/> Name	Instance ID
<input checked="" type="checkbox"/> Running	Status check
<input checked="" type="checkbox"/> t3.micro	Alarm status
	Availability Zone
	Launch instances

Below the table, there is a single row of data:

<input checked="" type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input checked="" type="checkbox"/>	-	i-0d0cb5dd98173f60c	Running	t3.micro	○ 2/2 checks passed	No alarms	us-west-2a

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

The screenshot shows the AWS Amazon S3 Buckets page. A modal dialog box is open at the top left, displaying an error message: "You don't have permissions to list buckets". The message states: "After you or your AWS administrator have updated your permissions to allow the s3:ListAllMyBuckets action, refresh this page. Learn more about Identity and access management in Amazon S3".

Name	AWS Region	Access	Creation date
Buckets (0)			

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

The screenshot shows the AWS EC2 Instances page. A prominent green banner at the top indicates that an instance has been successfully stopped. The main table lists one instance: 'i-0d0cb5dd98173f60c' which is currently 'Stopping'. The instance type is 't3.micro', it has passed 2/2 checks, and there are no alarms. It is located in the 'us-west-2a' availability zone. The left sidebar shows navigation options like EC2 Dashboard, Events, Tags, Limits, Instances (selected), and Images.

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

The screenshot shows a Gmail inbox with one message from 'Ruchika S <ms.ruchika23@gmail.com>'. The subject of the email is 'You finished a Qwiklab'. The email body starts with 'Hi Ruchika Shashidhara,' and informs the recipient that they completed the lab 'Introduction to AWS Identity and Access Management (IAM)'. It states that their completion percentage was 100.0% and encourages them to visit their dashboard. The email ends with a recommendation for another lab: 'Applied Machine Learning: Building Models for an Amazon Use Case'. The Qwiklabs logo is visible at the bottom of the email.

