Netaji Subhas University of Technology

Cloud Computing

Practical File

EICSC18

Submitted To
Dr. Pinaki Chakraborty

Submitted By
Harshit Sharda
2022UEI2816
EIOT (3rd Year)

INDEX

Experiment	Practical	Page
1.	Setup a Virtual Machine using VMware or Virtual Box.	3
2.	Implement Seamless File transfer between virtual machines and host machine.	5
3.	Install C Compiler and Execute Programs in Virtual Machine.	7
4.	Create a web application that can be hosted on cloud.	9
5.	Launch Web Applications with help of a cloud platform Launcher.	11
6.	Simulate Cloud Scenario with CloudSim	13
7.	Implementation of Docker our machine	15
8.	Launch virtual machine using trystack (Online Openstack Demo Version)	17
9.	To simulate a basic cloud server that processes incoming tasks from users.	19

10.	Implement a Basic REST API	22
11.	Set Up an S3 Bucket	23
12.	Create a Simple Lambda Function	24
13.	Launch a Basic EC2 Instance	25
14.	Create and Attach an EBS Volume	26

ASSIGNMENT 1

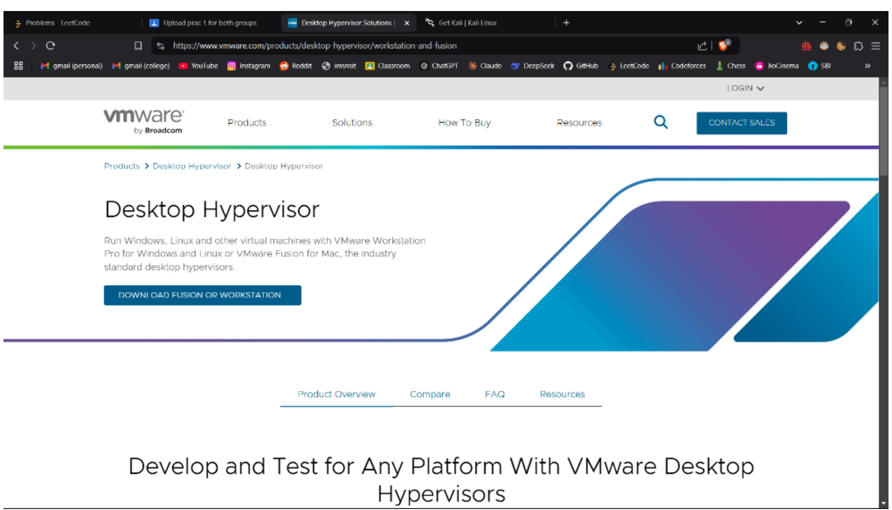
Aim: Setup Virtual Machine using VMware or Virtual Box

Procedure:

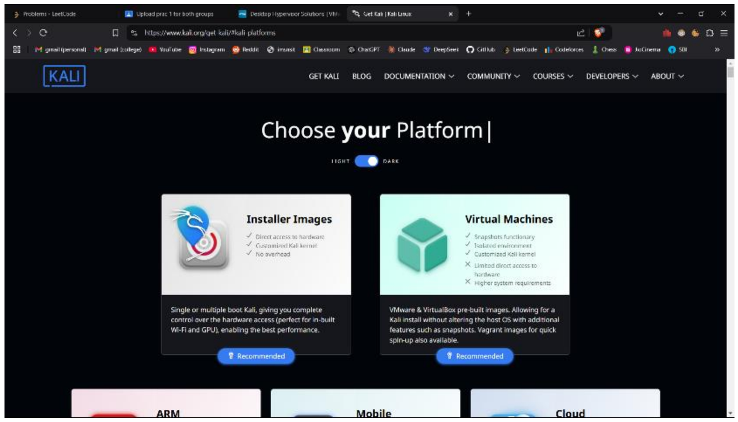
i) Ensure Virtualization is enabled in the task manager:

Base speed:	1.30 GHz
Sockets:	1
Cores:	10
Logical processors:	12
Virtualisation:	Enabled
L1 cache:	928 KB
L2 cache:	6.5 MB
L3 cache:	12.0 MB

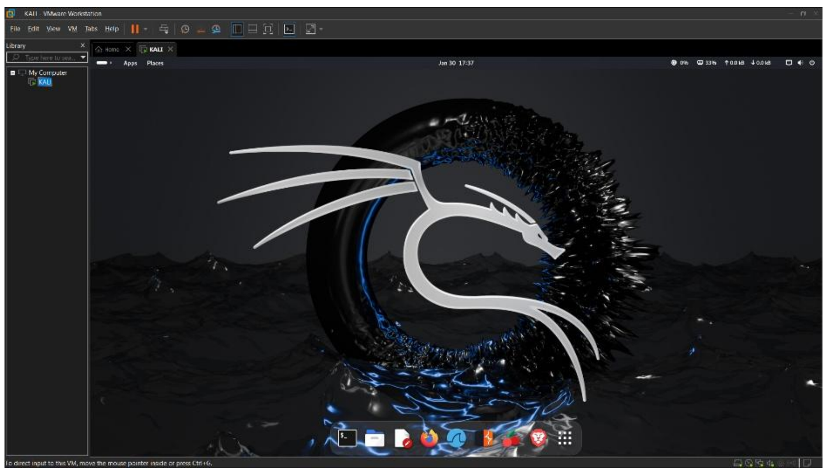
ii) Install VMware Workstation from official website: https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion

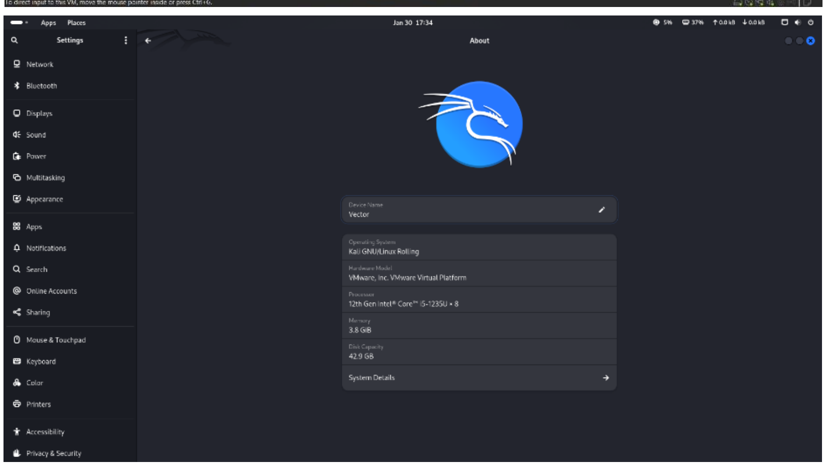


iii) Download installer image for the required OS (Kali Linux in this case) from official site: https://www.kali.org/get-kali/#kali-platforms



iv) Setup the hardware requirements and then setup the Virtual Machine





Result: All the steps were followed and a Kali Linux VM was setup on VMware.

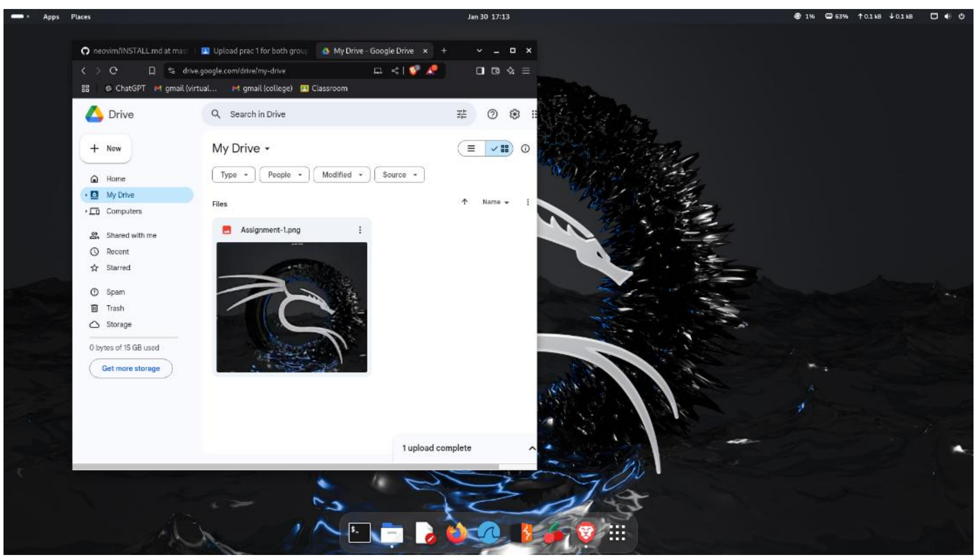
ASSIGNMENT 2

Aim: File transfer between virtual machines

- Explore procedures to transfer files between virtual machines using shared folders, network file transfer protocols (e.g. Google drive, Dropbox).
- Implement the chosen file transfer method to move files seamlessly between the virtual machines.

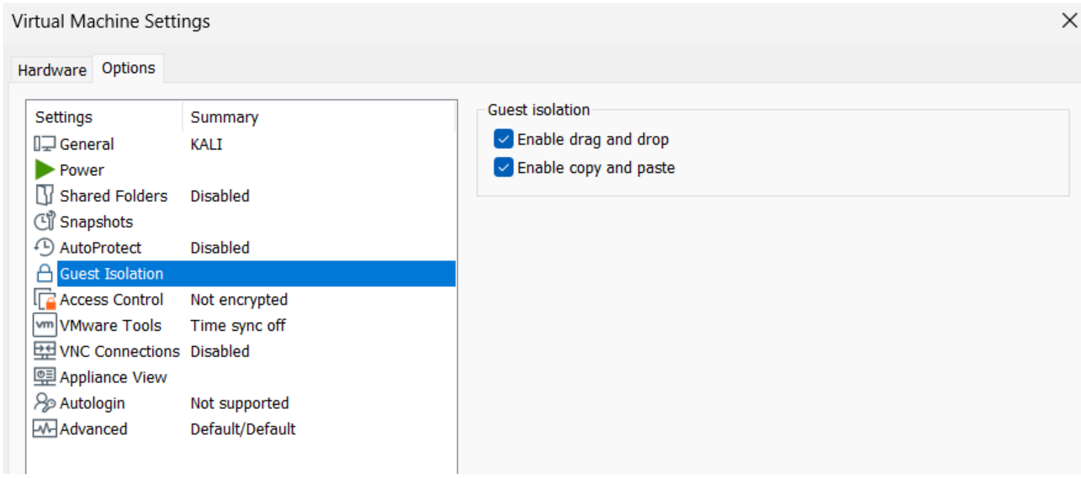
Procedure:

i) Login to a common g-mail in both host machine and virtual machine and open google drive



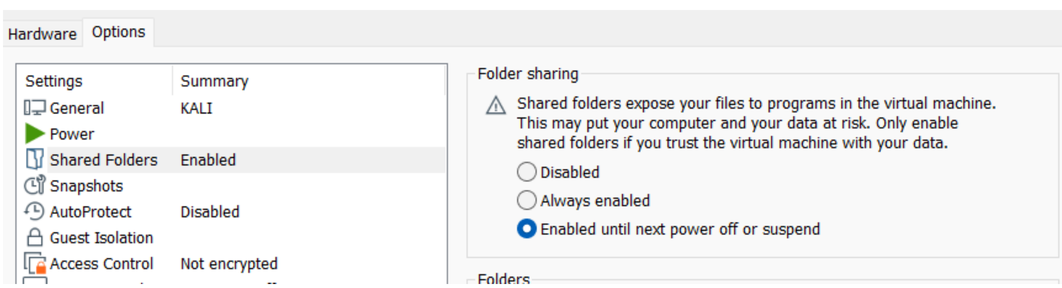
ii) Enable Bidirectional file transfer in VMware:

- Right click on VM and click settings.
- Got to options tab and select guest isolation
- Enable both options



iii) Enable shared Files if you want to access all the files from 1 machine to another

- Right click on VM and click settings.
- Got to options tab and select Shared Folders
- Enable for a session or keep always enabled.



Result: All the steps were followed and seamless file sharing was enabled between host machine and virtual machine.

ASSIGNMENT 3

Aim: Install C Compiler and Execute Programs:

• Within the virtual machine created using VirtualBox, install a C compiler (e.g., gcc) along with its dependencies.
• Write and compile simple C programs (e.g., Hello World, basic arithmetic) to ensure the compiler is functioning correctly.
• Execute the compiled programs to verify their output.

Procedure:

i) In Kali Linux gcc/g++ come pre-installed

```
┌──(vector㉿Vector)-[~/Desktop]
└─$ gcc --version
gcc (Debian 14.2.0-12) 14.2.0
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

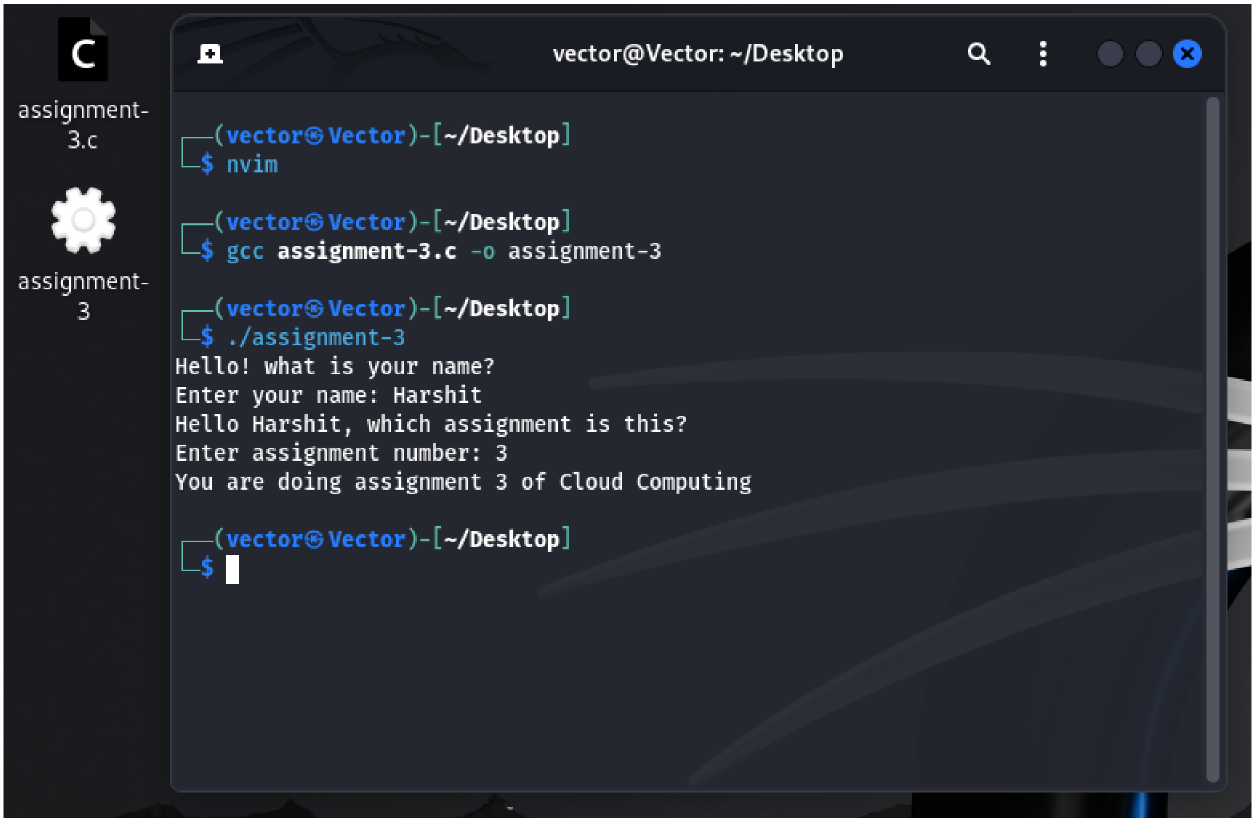
ii) Create a Simple code to implement basic IO functions

assignment-3.c

vector@Vector: ~/Desktop

```
#include<stdio.h>

int main () {
        printf("Hello! what is your name? \n");
        printf("Enter your name: ");

        char name[100];
        scanf("%s", name);

        printf("Hello %s, which assignment is this?\n", name);
        printf("Enter assignment number: ");

        int number;
        scanf("%d", &number);

        printf("You are doing assignment %d of Cloud Computing", number);

        return 0;
}
~
~
~
assignment-3.c                                          19,1          All
"assignment-3.c" 19L, 356B written
```

iii) Compile the code



Result: All the steps were followed and C program was compiled and executed successfully.

ASSIGNMENT 4

Aim: Install Google App Engine and Create Web Applications:
• Set up Google App Engine (GAE) development environment on your host
machine.
• Install the required development kits and frameworks for Python and Java.
• Create a simple "Hello World" web application using Python and Java for GAE.

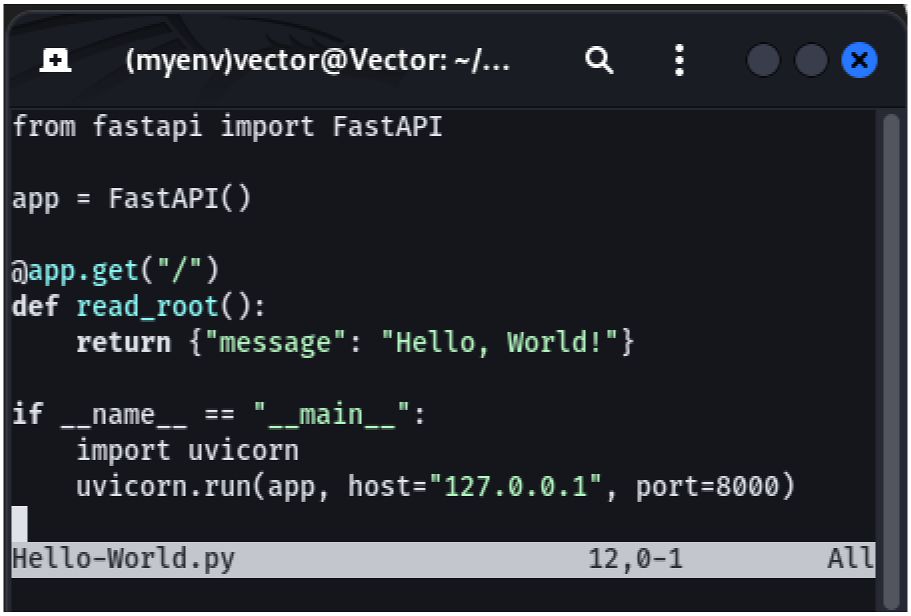
Procedure:

i) Create a virtual environment for development and install an application to host web Application, FastAPI in this case

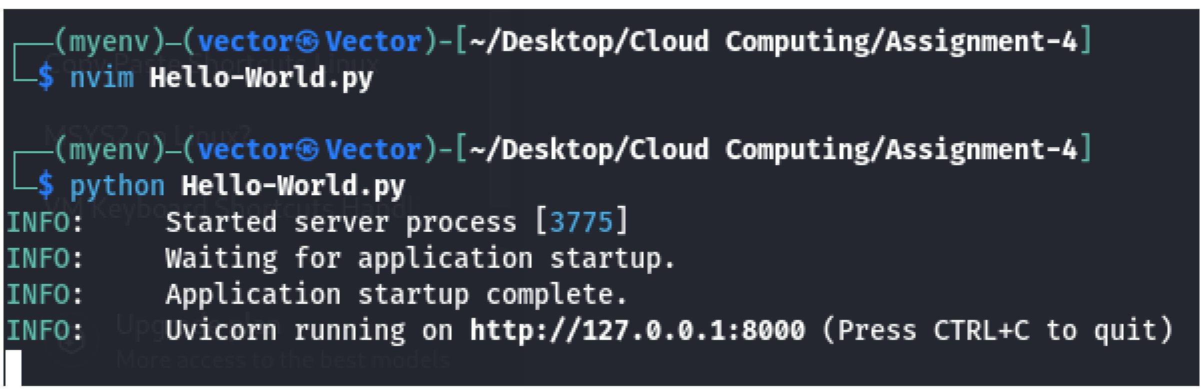
```
┌──(vector㉿Vector)-[~/Desktop/Cloud Computing/Assignment-4]
└─$ python3 -m venv myenv

┌──(vector㉿Vector)-[~/Desktop/Cloud Computing/Assignment-4]
└─$ source myenv/bin/activate

┌──(myenv)─(vector㉿Vector)-[~/Desktop/Cloud Computing/Assignment-4]
└─$ pip install fastapi uvicorn
Collecting fastapi
  Downloading fastapi-0.115.8-py3-none-any.whl.metadata (27 kB)
Collecting uvicorn
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
```

```
┌──(myenv)─(vector㉿Vector)-[~/Desktop/Cloud Computing/Assignment-4]
└─$ pip show fastapi
Name: fastapi
Version: 0.115.8
Summary: FastAPI framework, high performance, easy to learn, fast to code, ready for production
Home-page: https://github.com/fastapi/fastapi
Author:
```

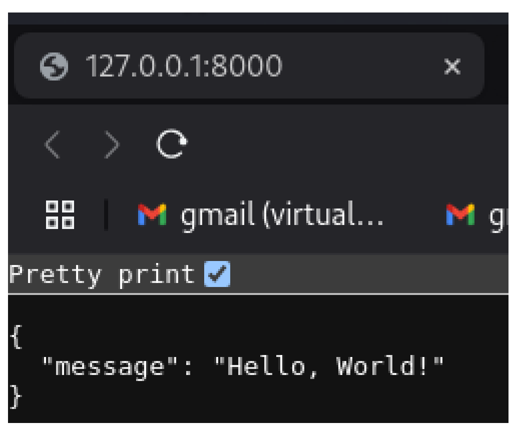
ii) Create a Simple code that hosts a page printing “Hello World”



iii) Start the server



iv) Open the port



Result: All the steps were followed and an environment was created to host a web application using FastAPI.

ASSIGNMENT 5

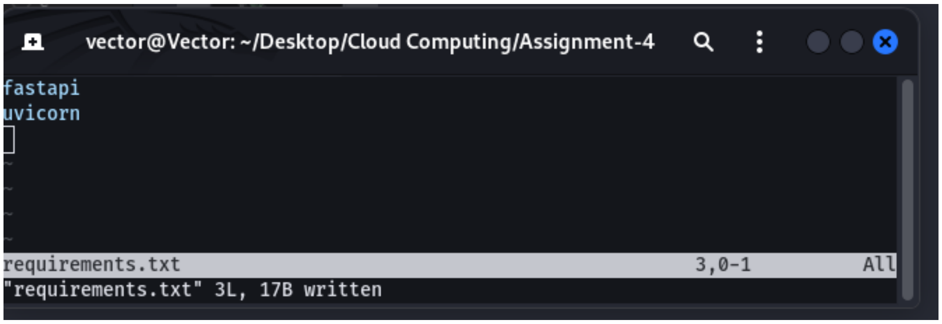
Aim: Launch Web Applications with Google App Engine Launcher:
• Use Google App Engine Launcher to deploy the web applications on Google's
cloud infrastructure.
• Launch the "Hello World" web application and other simple web apps to
experience the deployment process.

Procedure:

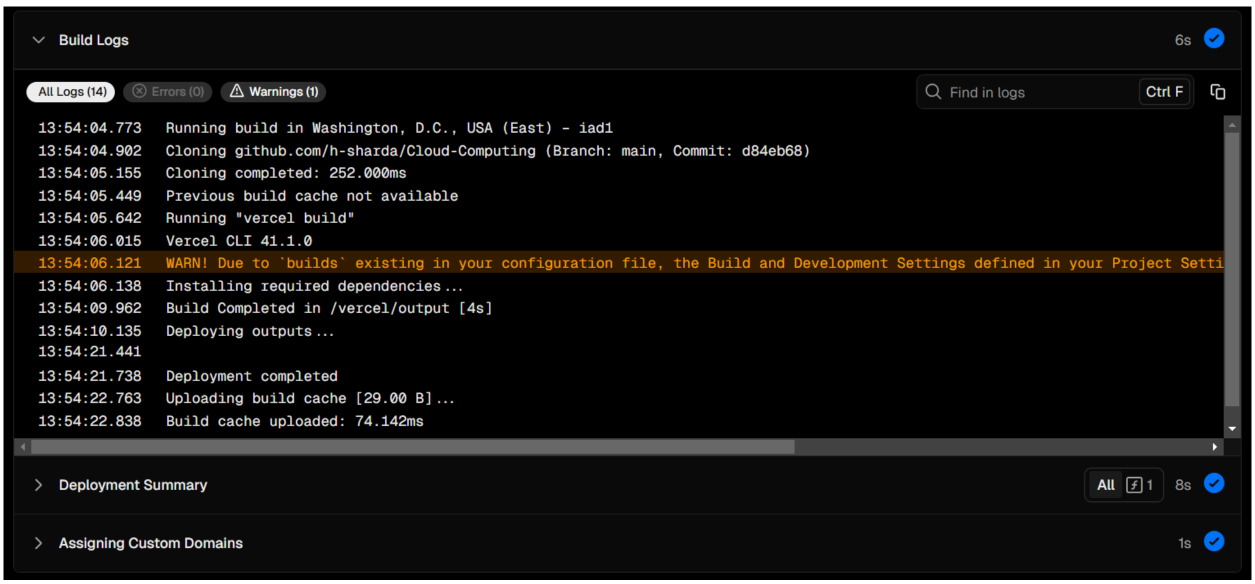
i) Web app was created in last assignment, here now will deploy it. First we need to choose a hosting platform (vercel in this case) so we need to set up configurations file for vercel.

```
{
        "builds": [
            {
                "src": "main.py",
                "use": "@vercel/python"
            }
        ],
        "routes": [
            {
                "src": "/(.*)",
                "dest": "main.py"
            }
        ]
}
~
~
~
~
~
~
~
~
vercel.json                                                  1,1           All
```

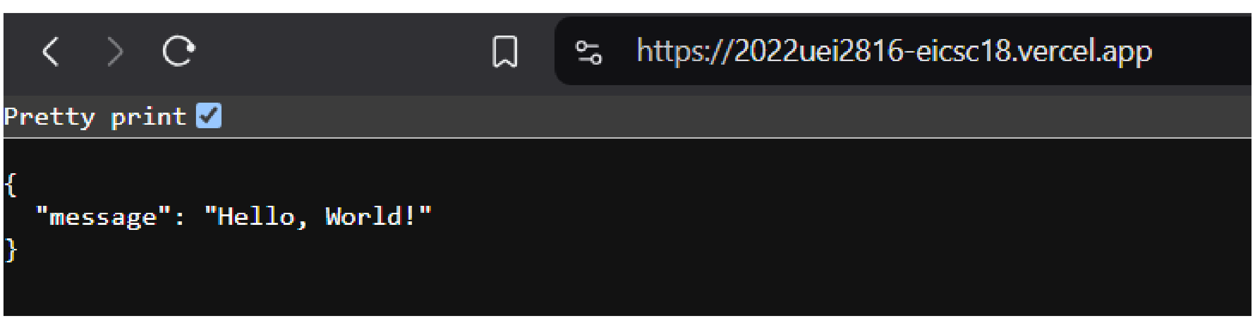
ii) Now we need a requirements file for the python web application .



iii) Now we deploy the web app. LOGS:



iv) Open the assigned URL:



Result: All the steps were followed and web app was successfully hosted on vercel. URL: https://2022uei2816-eicsc18.vercel.app/

ASSIGNMENT 6

Aim: Simulate Cloud Scenario with CloudSim:
• Install CloudSim, a cloud computing simulation toolkit, on your host machine or within a virtual machine.
• Set up a cloud scenario with virtual machines, data centers, and cloudlets.
• Implement a custom scheduling algorithm (not present in CloudSim) to allocate resources efficiently and run it in the simulated cloud environment

Procedure:

1. Download CloudSim from CloudSim GitHub Repository.
2. Install a Java IDE (e.g., Eclipse) and configure it with the CloudSim library.
3. Create a Java class and import CloudSim packages.
4. Define data centers, virtual machines (VMs), cloudlets, and brokers.
5. Write a custom scheduling algorithm (e.g., Round Robin or Weighted Fair Queueing).
6. Compile and run the simulation.

```
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
400.1: Broker: Cloudlet 0 received
400.1: Broker: All Cloudlets executed. Finishing...
400.1: Broker: Destroying VM #0
Broker is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0          SUCCESS        2              0       400       0.1          400.1
CloudSimExample1 finished!
```

Result: A basic cloud environment with VMs and cloudlets was successfully simulated using CloudSim, and a custom scheduling strategy was tested.

ASSIGNMENT 7

Aim: Docker Lab Task
1. Install Docker on your machine and verify the installation with docker --version.
2.Run your first container by executing docker run hello-world.
3. Create a simple Node.js/python web app that serves "Hello World!".
4. Build a Docker image for your app using a Dockerfile and run it with port mapping.
5. Deploy your app to a Docker Swarm using a docker-compose.yml file and access it in your browser.

Procedure:

1. Install Docker using sudo apt install docker.io (Linux) or download from docker.com.
2. Verify installation: docker –version
3. Run the first container: docker run hello-world
4. Create a simple app and a docker file and run container

```
C:\Windows\System32>docker --version
Docker version 28.0.1, build 068a01e

C:\Windows\System32>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/
```

```
D:\College\Sem 6\Cloud Computing\Lab\exp-7>docker stack deploy -c docker-compose.yml my-app
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network my-app_default
Creating service my-app_web

D:\College\Sem 6\Cloud Computing\Lab\exp-7>docker service ls
ID             NAME         MODE         REPLICAS   IMAGE                 PORTS
vjhot7upienn   my-app_web   replicated   0/2        my-web-app:latest     *:3000->3000/tcp

D:\College\Sem 6\Cloud Computing\Lab\exp-7>
```

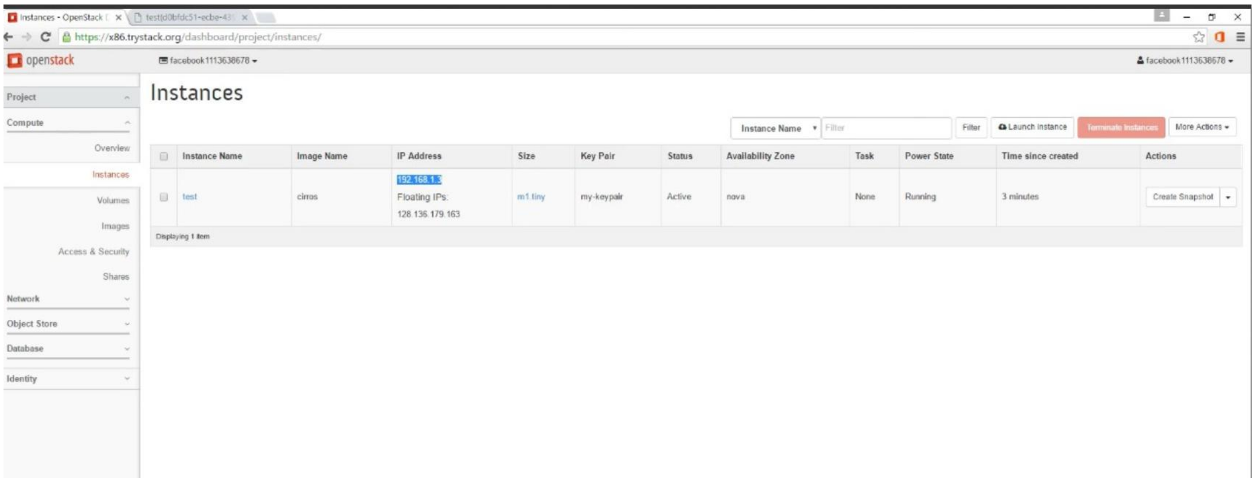
Result: Docker was successfully installed. A containerized Python web app was built and accessed on localhost:5000.

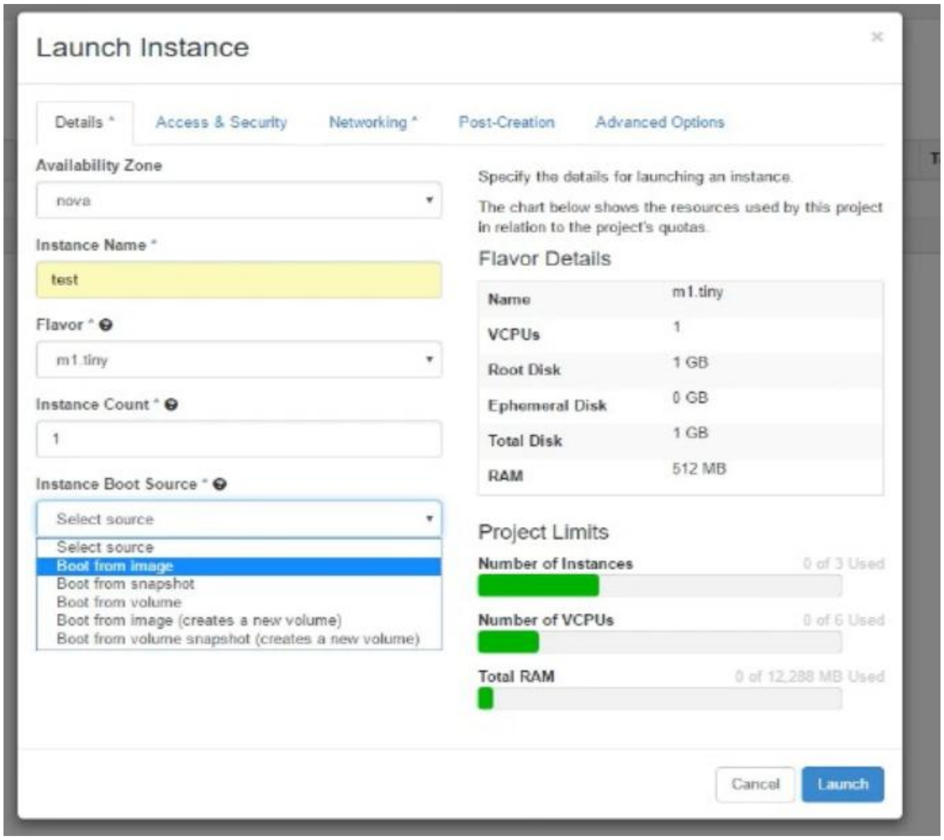
ASSIGNMENT 8

Aim: Launch virtual machine using trystack(Online Openstack Demo Version).

Procedure:

1. Visit: https://trystack.org
2. Log in using Facebook to access the dashboard.
3. Select a flavor (RAM, CPU, storage) and OS image (Ubuntu, CentOS).
4. Launch a new instance with key pair for SSH access.
5. Access terminal console or connect using SSH.





Result: A virtual machine was successfully launched and accessed using TryStack's OpenStack environment.

ASSIGNMENT 9

Aim: Cloud Server Task Processing Simulation
Objective:
To simulate a basic cloud server that processes incoming tasks from users, queues them if the server
is busy, and studies the effect of task arrival rates on server performance.

1. Introduction
Cloud servers process multiple user requests. In this lab, we will use AnyLogic to model a simple
cloud task processing system where:
 Users generate tasks at a specific rate.
 A cloud server processes tasks one at a time.
 Tasks wait in a queue if the server is busy.
 We analyze how the queue length changes with different loads.

2. Tools & Software Required
 AnyLogic Personal Learning Edition (Download from AnyLogic)
 Basic understanding of agent-based modeling and process flow simulation.

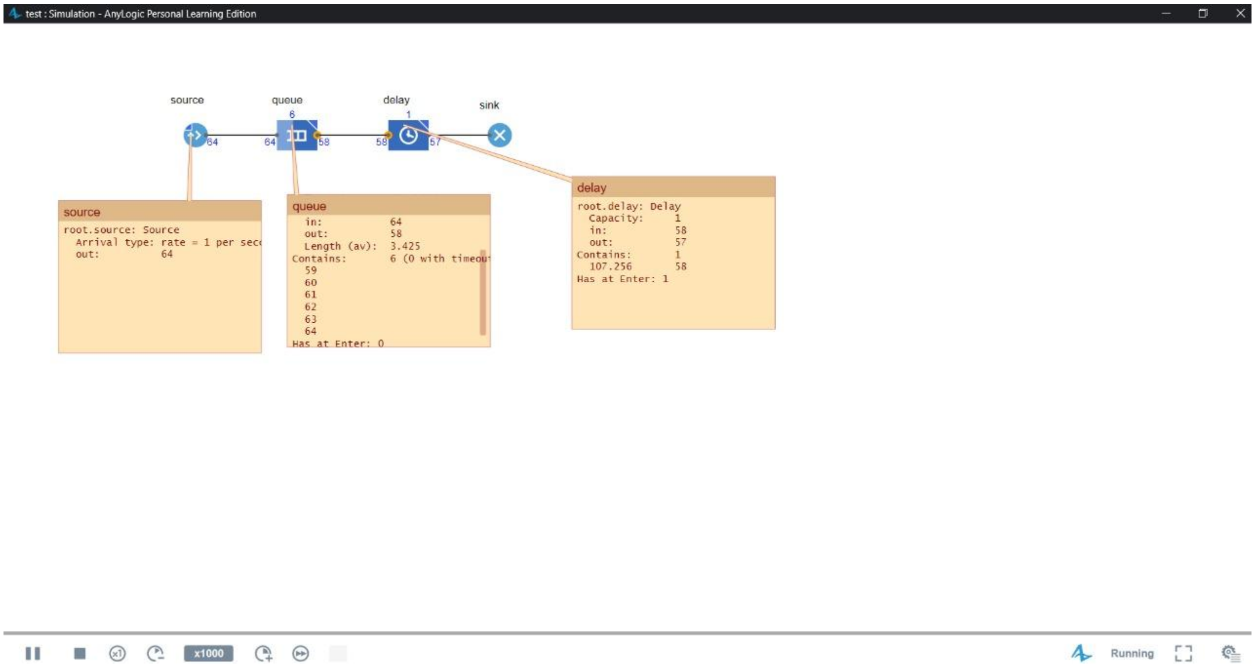
3. Steps to Implement the Model
Step 1: Create a New AnyLogic Model
1. Open AnyLogic and create a new project.
2. Set the Main agent as CloudServer.

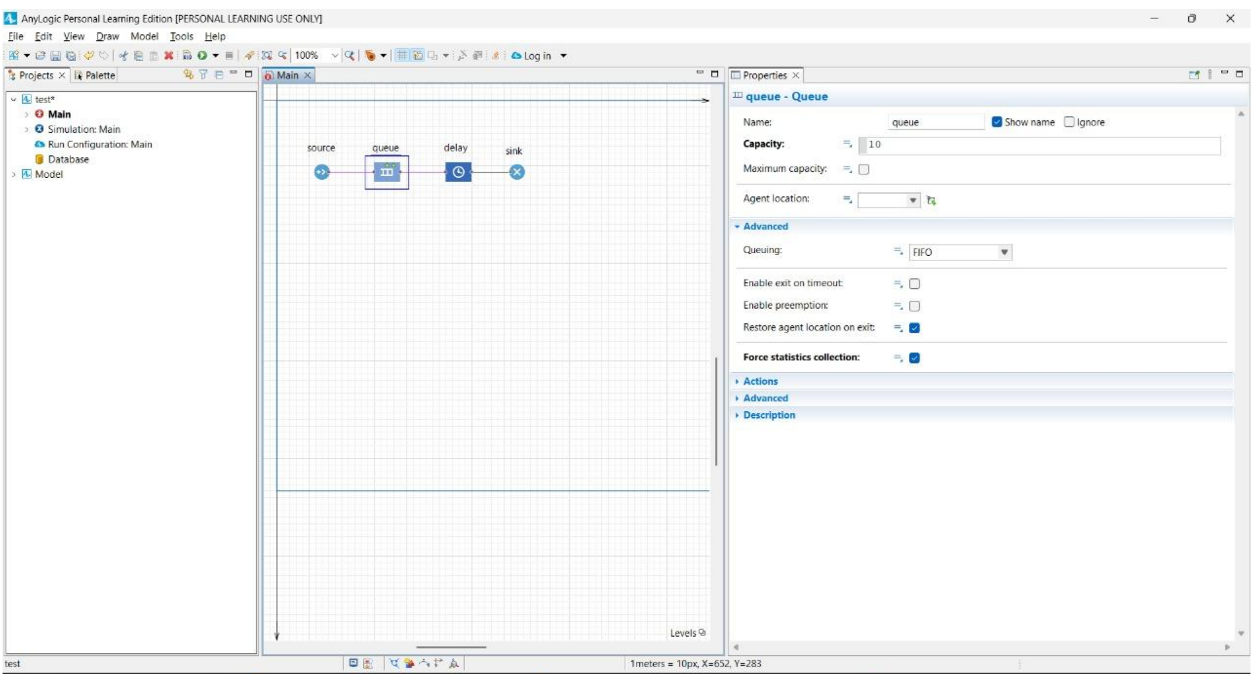
Step 2: Define the Task Flow Using the Process Modeling Library
Drag and drop the following blocks from the Process Modeling Library:
Block Purpose
Sourc
e
Generates user requests (tasks) at regular
intervals.
Queue Holds tasks when the server is busy.

Delay Represents the server processing the task.
Sink Represents task completion (exit from the system).
Configure Each Block:

1. Source (Task Generator)
o Arrival rate: One task every 5 seconds.
o Entity type: Task.
2. Queue (Task Waiting Area)
o Capacity: 10 (limited queue size to simulate congestion).
3. Delay (Cloud Server Processing Time)
o Random processing time: Between 3 to 7 seconds (Uniform distribution).
4. Sink (Task Completion)
o Tasks leave after processing.

Step 3: Run and Observe the Simulation
1. Start the simulation and observe:
o How tasks enter the system.
o If the queue overflows when the server is slow.
o Whether processing time impacts performance.
2. Modify parameters:
o Increase task arrival rate (one task every 2 seconds).
o Observe how the queue fills up.





Learning Outcomes

Understand task scheduling and queueing in cloud computing.
Learn how server processing speed affects system performance.
Explore bottlenecks and ways to optimize cloud workloads.

ASSIGNMENT 10

Aim: Implement a basic REST API

Procedure:

1. Install FastAPI and Uvicorn:

pip install fastapi uvicorn

2. Create main.py:

```javascript
async function callApi(url, method = 'GET', data = null, headers = {}) {
  try {
    const options = {
      method,
      headers: {
        'Content-Type': 'application/json',
        ...headers,
      },
    };

    if (data) {
      options.body = JSON.stringify(data);
    }

    const response = await fetch(url, options);

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const result = await response.json();
    return result;
  } catch (error) {
    console.error('API call failed:', error);
    return null;
  }
}
```

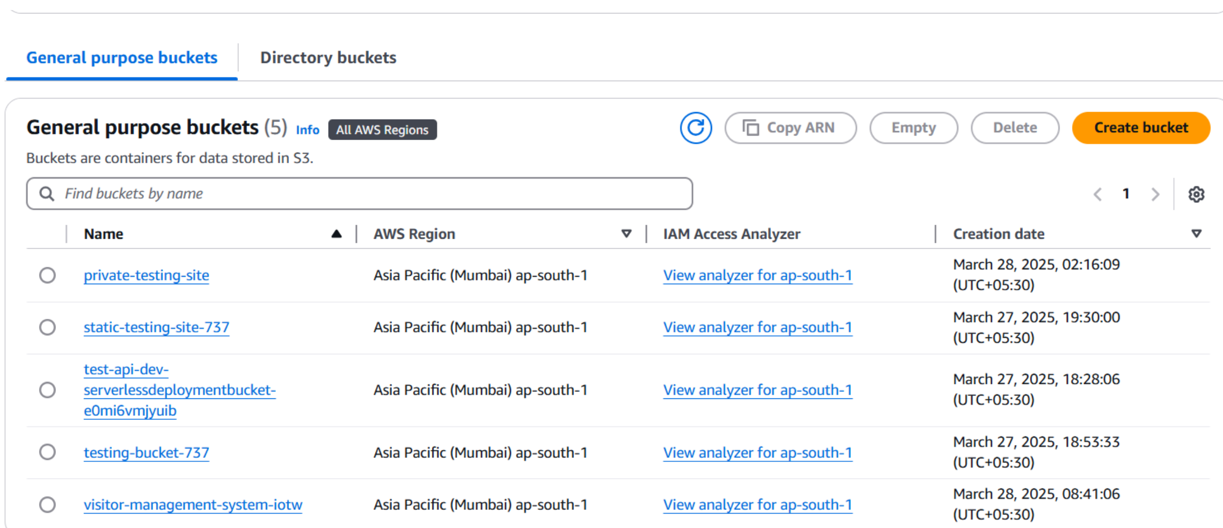
Result: A basic REST API was implemented and tested successfully on http://localhost:8000.

ASSIGNMENT 11

Aim: Set Up an S3 Bucket

Procedure:

1. Log in to AWS Console.
2. Go to **S3** and click **Create bucket**.
3. Enter a unique name, select region, and leave other settings default.
4. Upload a file to test.
5. Set permissions to public or private as required.



Result: An S3 bucket was successfully created, and files were uploaded and managed through the AWS Console.

ASSIGNMENT 12

Aim: Create a Simple Lambda Function

Procedure:

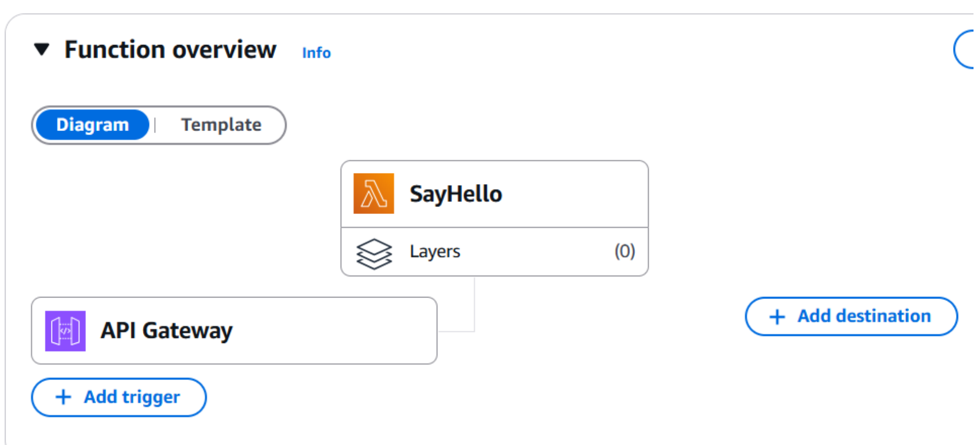
1. Go to AWS Lambda in AWS Console.
2. Click Create Function > Author from Scratch.
3. Use runtime: Python 3.9
4. Add function code:

JS index.mjs ×

JS index.mjs > ...

```
export const handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({message: 'Hello from Harshit\' AWS Lambda!'}),
  };
  return response;
};

```

5. Click Test with sample event.



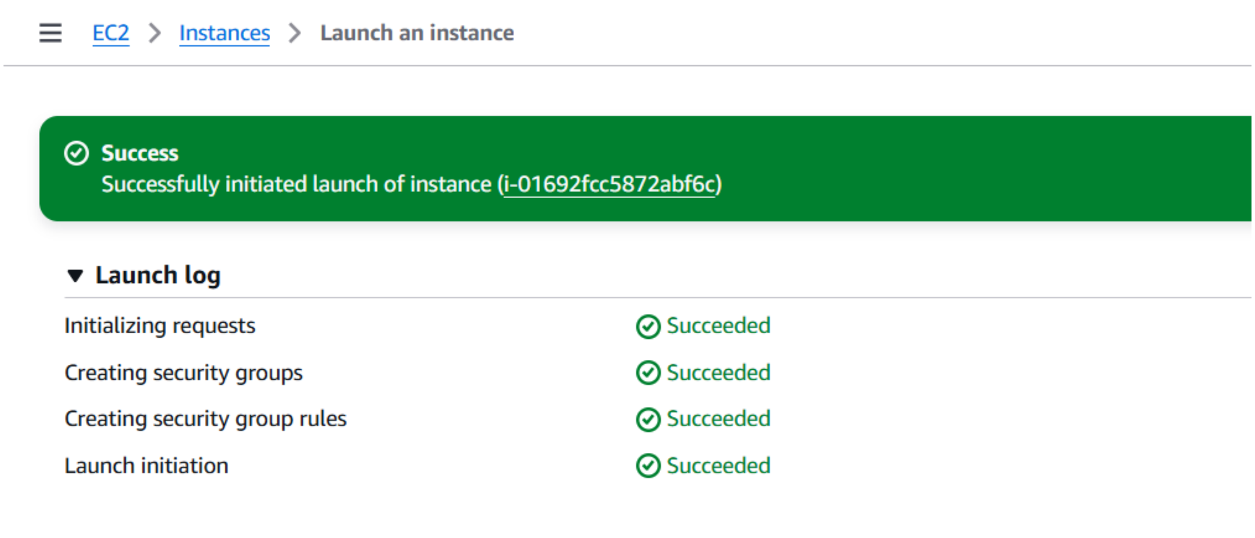
Result: Lambda function executed successfully and returned a test response.

ASSIGNMENT 13

Aim: Launch a Basic EC2 Instance

Procedure:

1. Go to **EC2 Dashboard** > Launch Instance.
2. Select AMI: Ubuntu Server.
3. Choose instance type: t2.micro.
4. Create key pair and download .pem file.
5. Configure security group to allow SSH (port 22).
6. Launch and connect using:



Result: EC2 instance was launched and accessed via SSH.

ASSIGNMENT 14

Aim: Create an Elastic Block Store (EBS) volume in AWS Educate and attach it to an existing EC2 instance. Format and mount the volume to be used as additional storage.

Procedure:

1. Go to EC2 > Elastic Block Store > Volumes.
2. Click **Create Volume**, select size and availability zone.
3. Attach to an existing EC2 instance.
4. SSH into the instance and run:

bash Copy Edit

```bash
lsblk
sudo mkfs.ext4 /dev/xvdf
sudo mkdir /ebs
sudo mount /dev/xvdf /ebs
```

Result: EBS volume was created, formatted, mounted, and used as additional storage in EC2 instance.