VCC Assignment 3
Report

Gourav Sen
M24CSA011

Github: Github link
Video: Video Link

---

1. Introduction

Objective: **Autoscaling a web application from a local VM to GCP** when resource usage reaches 75%, followed by live migration back to the local machine when load decreases.

- Tools and technologies used:
  - **VirtualBox** for local VM
  - **Lubuntu** as the guest OS
  - **Docker** for containerizing the application
  - **htop** for resource monitoring
  - **GCP** for autoscaling and live migration
  - **Bash scripts** for automation
- Architecture Diagram

---

2. Prerequisites and Setup

- **System Requirements**
  - Host Machine: OS, RAM, CPU, Disk space
  - Guest VM: Lubuntu configuration (RAM, CPU, Disk)
- **Downloading Required Software**
  - **VirtualBox**
  - **Lubuntu ISO**
  - **GCP SDK**
- **Installing VirtualBox**
- **Creating Lubuntu VM**
  - **VM Configuration**
  - **Network Configuration (NAT with Port Forwarding)**

---

3. VM Configuration and Access

- **Starting the Lubuntu VM**
- **Port Forwarding Rules**
  - SSH Access: Host Port 2222 → Guest Port 22
  - WebApp Access: Host Port 8080 → Guest Port 80
- **SSH into the VM from the Host**
  - Verifying connectivity
  - Troubleshooting SSH issues

---

4. Docker Installation and Setup

- **Installing Docker on Lubuntu**

- **Configuring Docker**
- **Verifying Docker Installation**
- **Installing Required Packages**
    - Python
    - Pip
    - Docker dependencies

---

5. Web Application Development

- **Creating app.py:**
    - Simple Python Flask-based web application
    - Application will stress the CPU to simulate high load
- **Creating requirements.txt:**
    - Define required Python packages
- **Building and Running the Docker Container**
- **Testing the Web Application**

---

6. Monitoring and Resource Management

- **Installing htop:**
    - Real-time resource usage monitoring
- **Creating the Dockerfile:**
    - Define environment and dependencies
- **Resource Monitoring Script:**
    - **Bash script** to monitor CPU usage
    - Trigger autoscaling when CPU > 75%

---

7. Autoscaling to GCP

- **Installing Google Cloud SDK**
    - Download and install GCP SDK
    - Authenticate using the **service account**
- **Service Account Setup**
    - Create and configure service account
    - Assign necessary IAM roles
- **GCP VM Instance Creation**
    - Launch VM on GCP
    - Assign static external IP
- **Load Transfer to GCP**
    - Redirect traffic from local VM to GCP

---

8. Live Migration and Load Balancing

- **Live Migration:**
    - Trigger GCP instance when CPU > 75%
    - Migrate back to local VM when load decreases
- **Automation Script:**
    - Bash script to automate the entire process

---

9. Results and Observations

- VM Creation and Deletion Logs

---

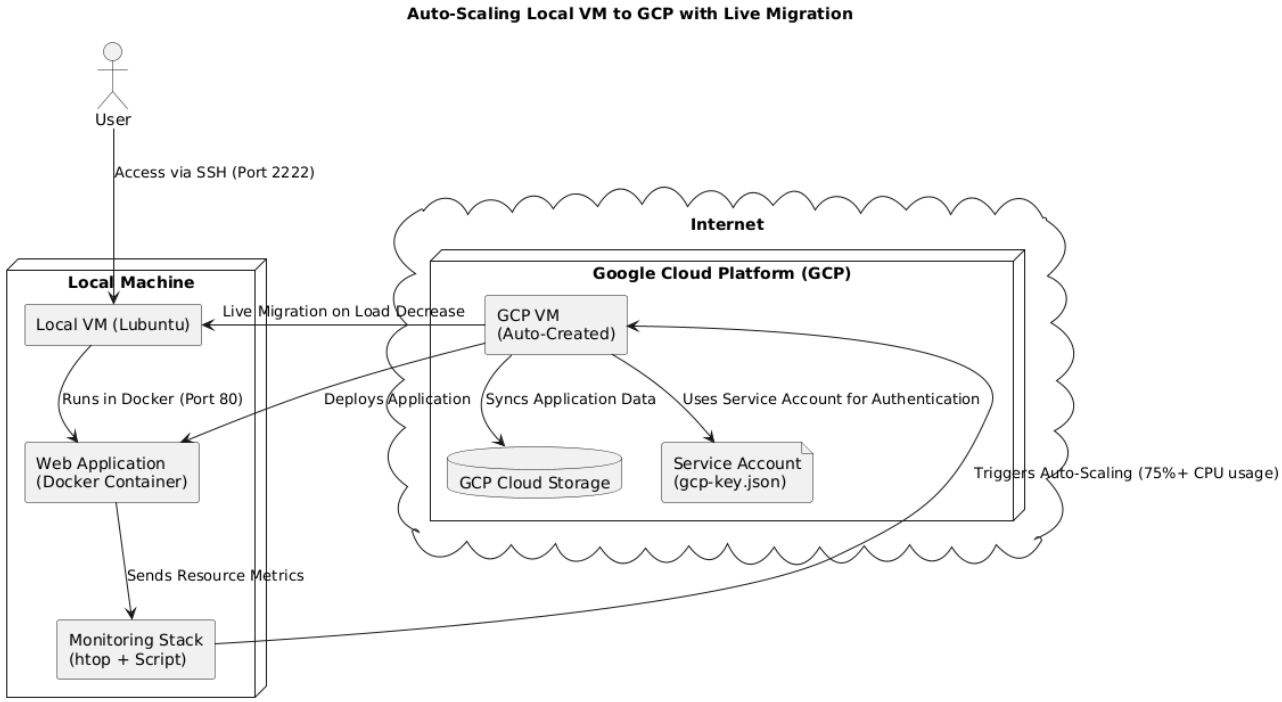
10. Conclusion

- Summary of the project
- Challenges faced
- Possible improvements

- # 1. Introduction

  The objective of this project is to demonstrate **autoscaling and live migration of a web application** from a local Virtual Machine (VM) to **Google Cloud Platform (GCP)** based on **resource usage**. The application, running on a **Lubuntu VM**, is monitored for CPU utilization. When the CPU usage exceeds **75%**, the application is automatically scaled to a **GCP instance**. When the load decreases, the application is migrated back to the local VM to **optimize resource usage**.

  ## Architecture Diagram

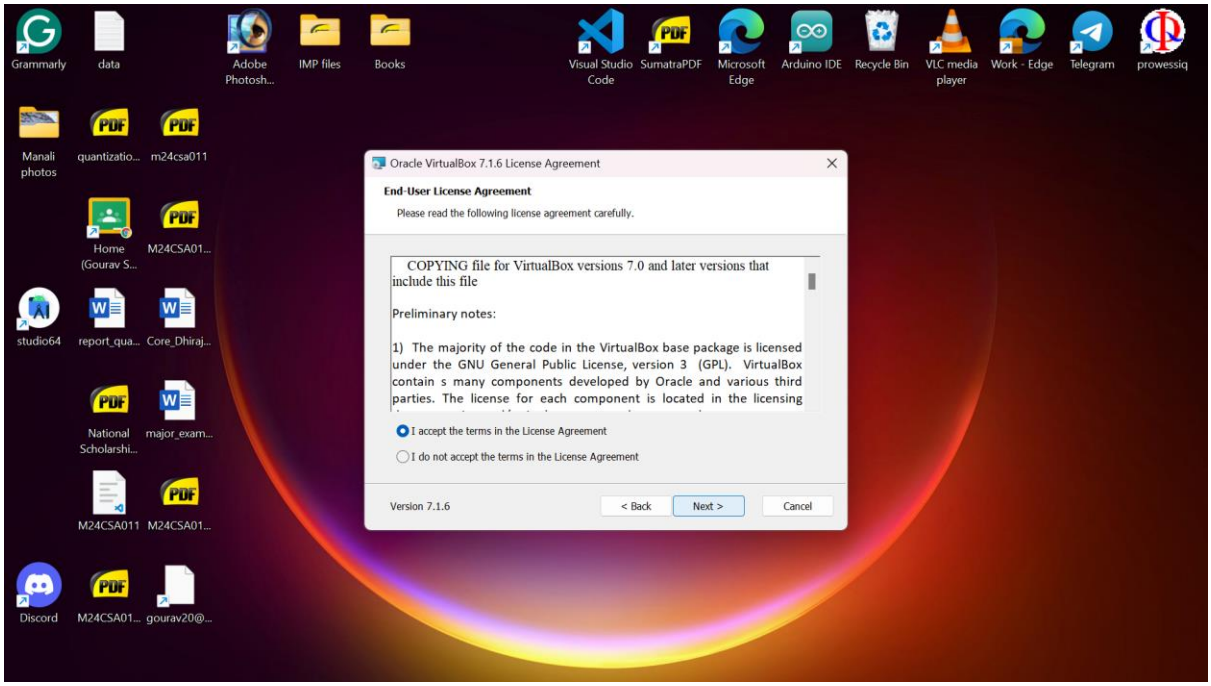


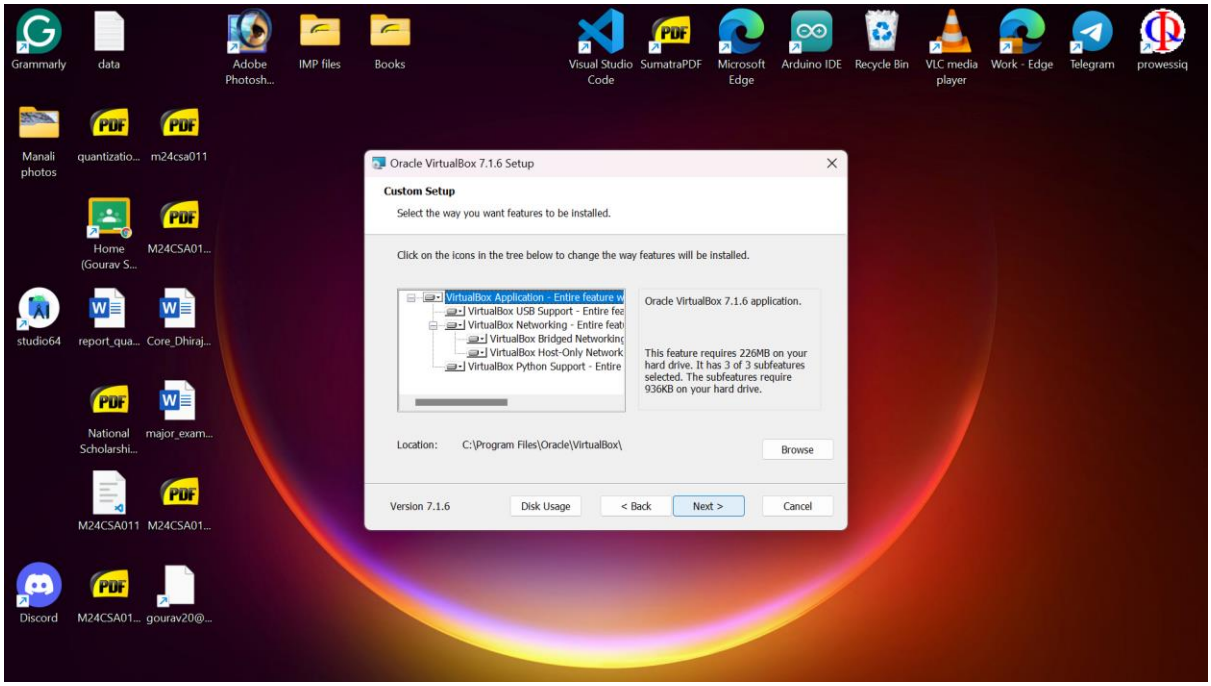
2. Prerequisites and Setup

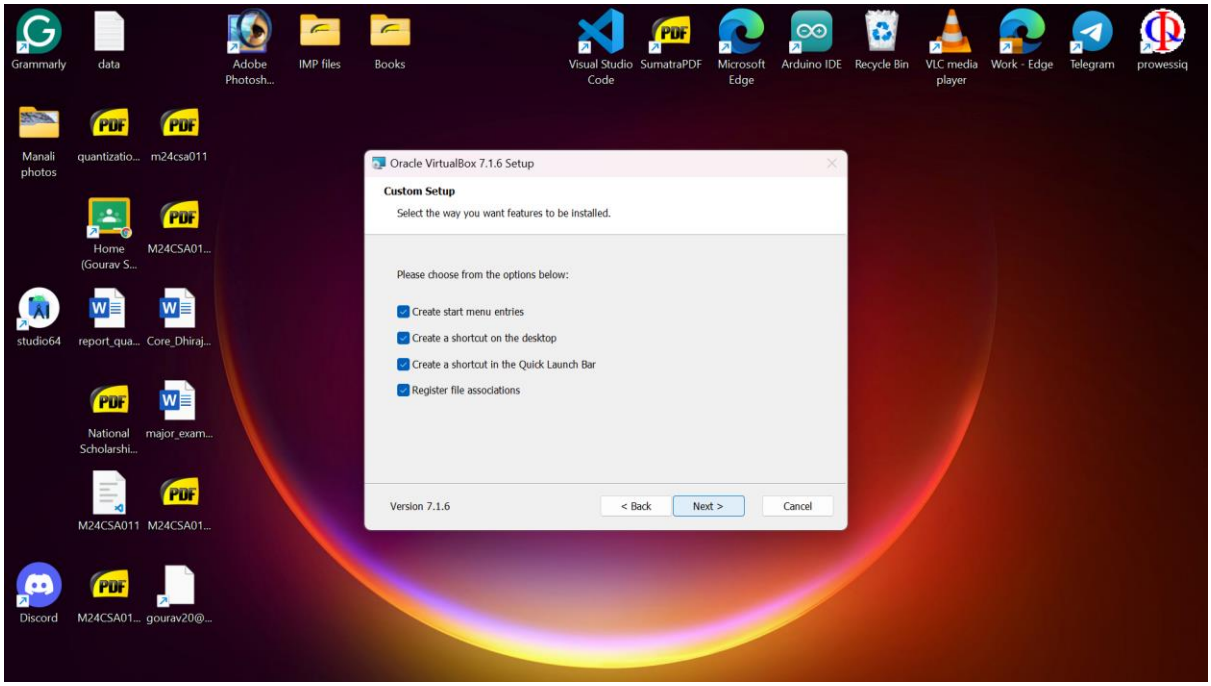
2.1 Install VirtualBox

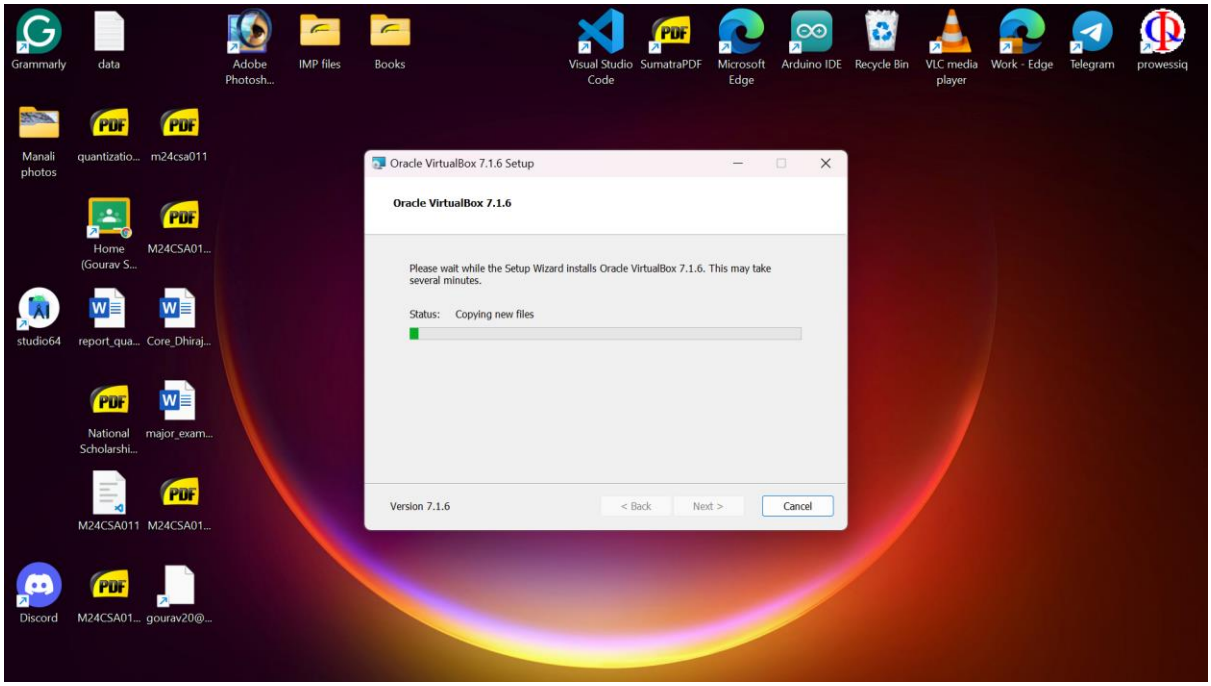
Download and install VirtualBox from the official website:
[VirtualBox Download]

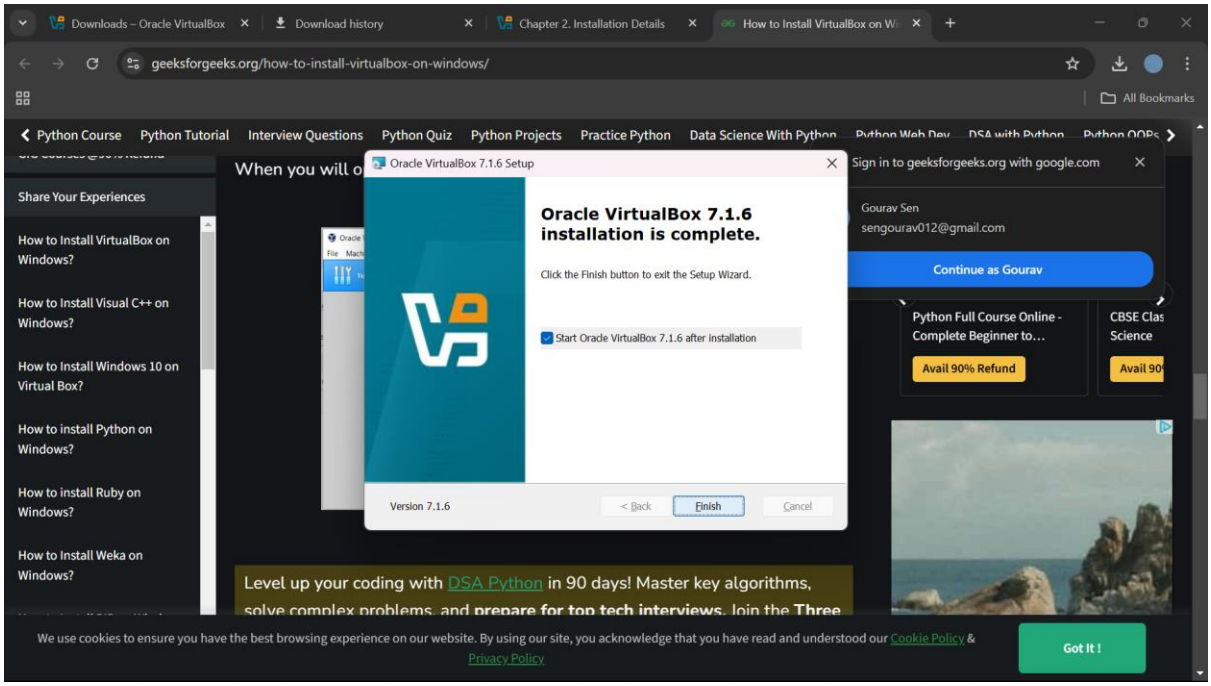

Grammarly
data
Adobe Photosh...
IMP files
Books
Visual Studio Code
SumatraPDF
Microsoft Edge
Arduino IDE
Recycle Bin
VLC media player
Work - Edge
Telegram
prowessiq
Manali photos
quantizatio...
m24csa011
Home (Gourav S...
M24CSA01...
studio64
report_qua...
Core_Dhiraj...
National Scholarshi...
major_exam...
M24CSA011
M24CSA01...
Discord
M24CSA01...
gourav20@...
Oracle VirtualBox 7.1.6 Setup
Welcome to the Oracle VirtualBox 7.1.6 Setup Wizard
The Setup Wizard will install Oracle VirtualBox 7.1.6 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Version 7.1.6
Next >
Cancel


Grammarly
data
Adobe Photosh...
IMP files
Books
Visual Studio Code
SumatraPDF
Microsoft Edge
Arduino IDE
Recycle Bin
VLC media player
Work - Edge
Telegram
prowessiq
Manali photos
quantizatio...
m24csa011
Home (Gourav S...
M24CSA01...
studio64
report_qua...
Core_Dhiraj...
National Scholarshi...
major_exam...
M24CSA011
M24CSA01...
Discord
M24CSA01...
gourav20@...
Oracle VirtualBox 7.1.6 License Agreement
End-User License Agreement
Please read the following license agreement carefully.
COPYING file for VirtualBox versions 7.0 and later versions that include this file
Preliminary notes:
1) The majority of the code in the VirtualBox base package is licensed under the GNU General Public License, version 3 (GPL). VirtualBox contain s many components developed by Oracle and various third parties. The license for each component is located in the licensing
I accept the terms in the License Agreement
I do not accept the terms in the License Agreement
Version 7.1.6
< Back
Next >
Cancel


Grammarly
data
Adobe Photosh...
IMP files
Books
Visual Studio Code
SumatraPDF
Microsoft Edge
Arduino IDE
Recycle Bin
VLC media player
Work - Edge
Telegram
prowessiq
Manali photos
quantizatio...
m24csa011
Home (Gourav S...
M24CSA01...
studio64
report_qua...
Core_Dhiraj...
National Scholarshi...
major_exam...
M24CSA011
M24CSA01...
Discord
M24CSA01...
gourav20@...
Oracle VirtualBox 7.1.6 Setup
Custom Setup
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.
VirtualBox Application - Entire feature w
VirtualBox USB Support - Entire fea
VirtualBox Networking - Entire feat
VirtualBox Bridged Networking
VirtualBox Host-Only Network
VirtualBox Python Support - Entire
Oracle VirtualBox 7.1.6 application.
This feature requires 226MB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 936KB on your hard drive.
Location: C:\Program Files\Oracle\VirtualBox\
Browse
Version 7.1.6
Disk Usage
< Back
Next >
Cancel


Grammarly
data
Adobe Photosh...
IMP files
Books
Visual Studio Code
SumatraPDF
Microsoft Edge
Arduino IDE
Recycle Bin
VLC media player
Work - Edge
Telegram
prowessiq
Manali photos
quantizatio...
m24csa011
Home (Gourav S...
M24CSA01...
studio64
report_qua...
Core_Dhiraj...
National Scholarshi...
major_exam...
M24CSA011
M24CSA01...
Discord
M24CSA01...
gourav20@...
Oracle VirtualBox 7.1.6 Setup
Custom Setup
Select the way you want features to be installed.
Please choose from the options below:
Create start menu entries
Create a shortcut on the desktop
Create a shortcut in the Quick Launch Bar
Register file associations
Version 7.1.6
< Back
Next >
Cancel





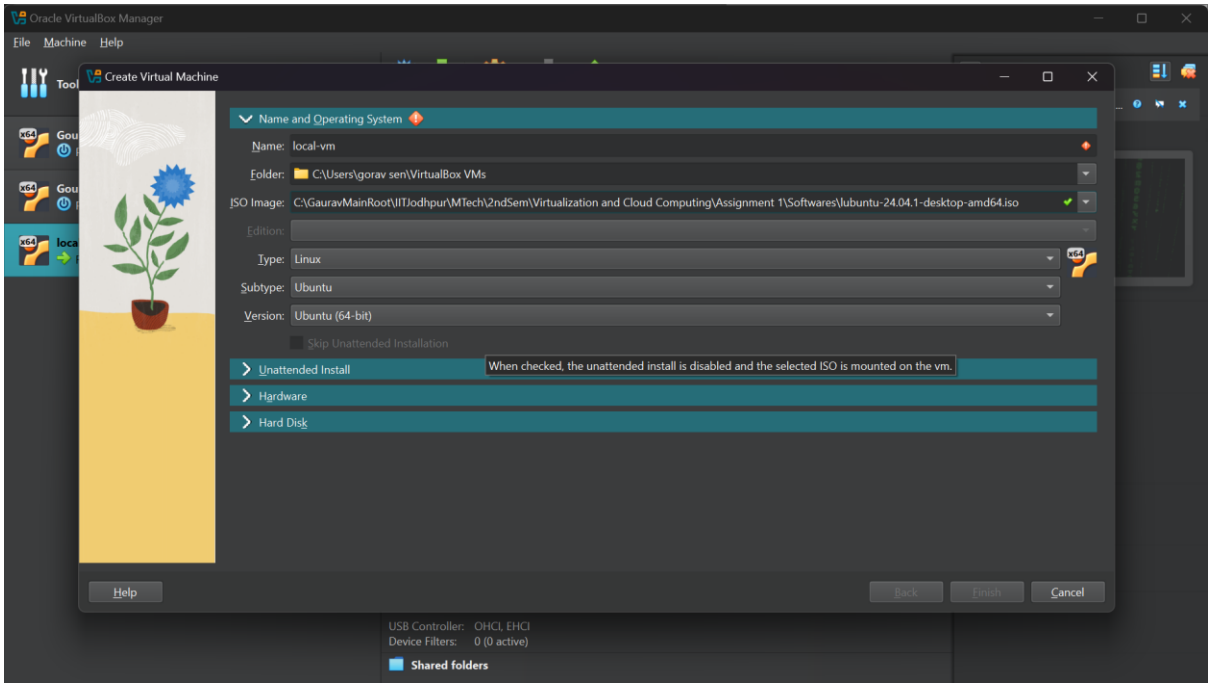
2.3 Download Lubuntu ISO

Since Ubuntu required more resources than available on our machine, we opted for Lubuntu, which is lightweight and suitable for systems with limited resources

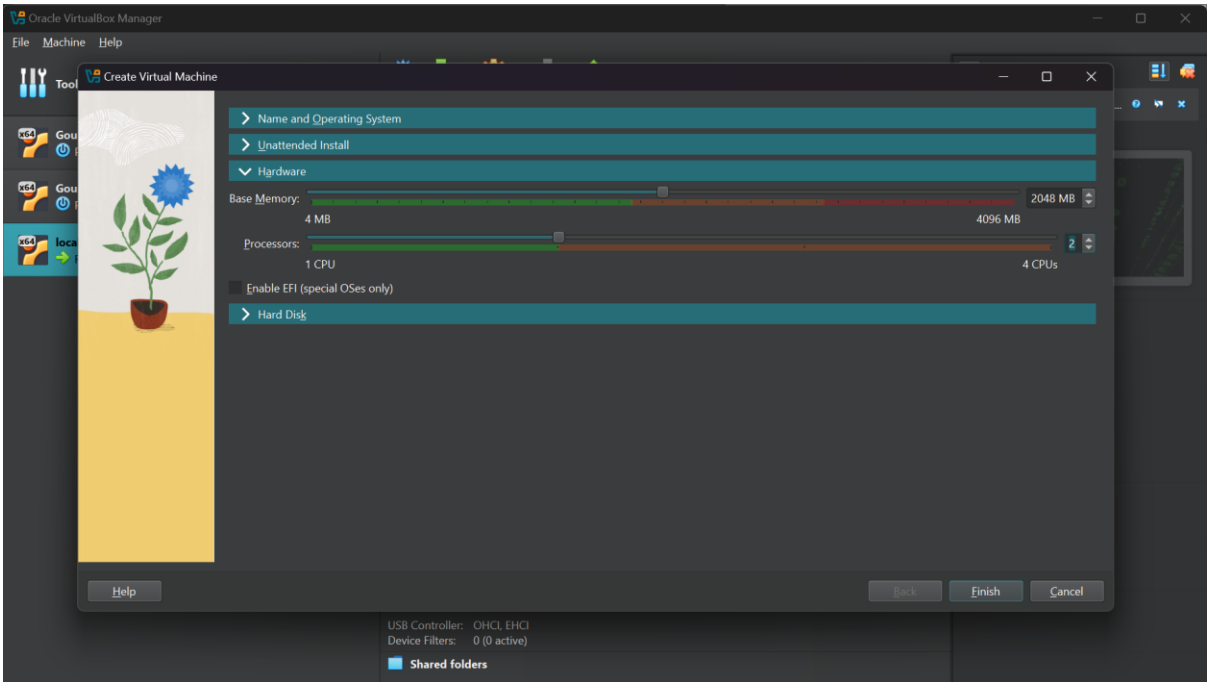
Download the **Lubuntu 22.04 ISO** from the official website:
Lubuntu Download

2.4 Create a New VM in VirtualBox

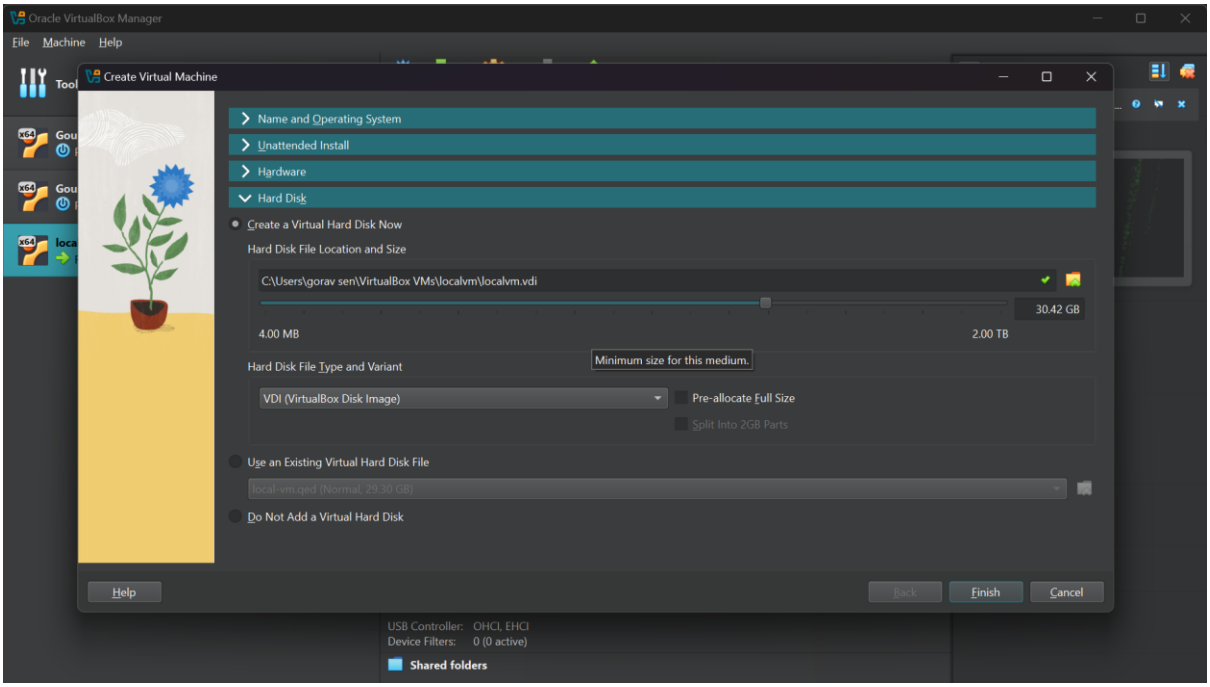
1. Open VirtualBox and click on **New**.
2. Set the following configurations:
    - **Name:** Lubuntu-VM
    - **Type:** Linux
    - **Version:** Ubuntu (64-bit)
    - **Memory Size:** 2 GB (2048 MB)
    - **Hard Disk:** Create a virtual hard disk now
    - **Disk Size:** 30 GB
3. Click **Create**.



Memory and processors:



Allocate hard disk and Finish:

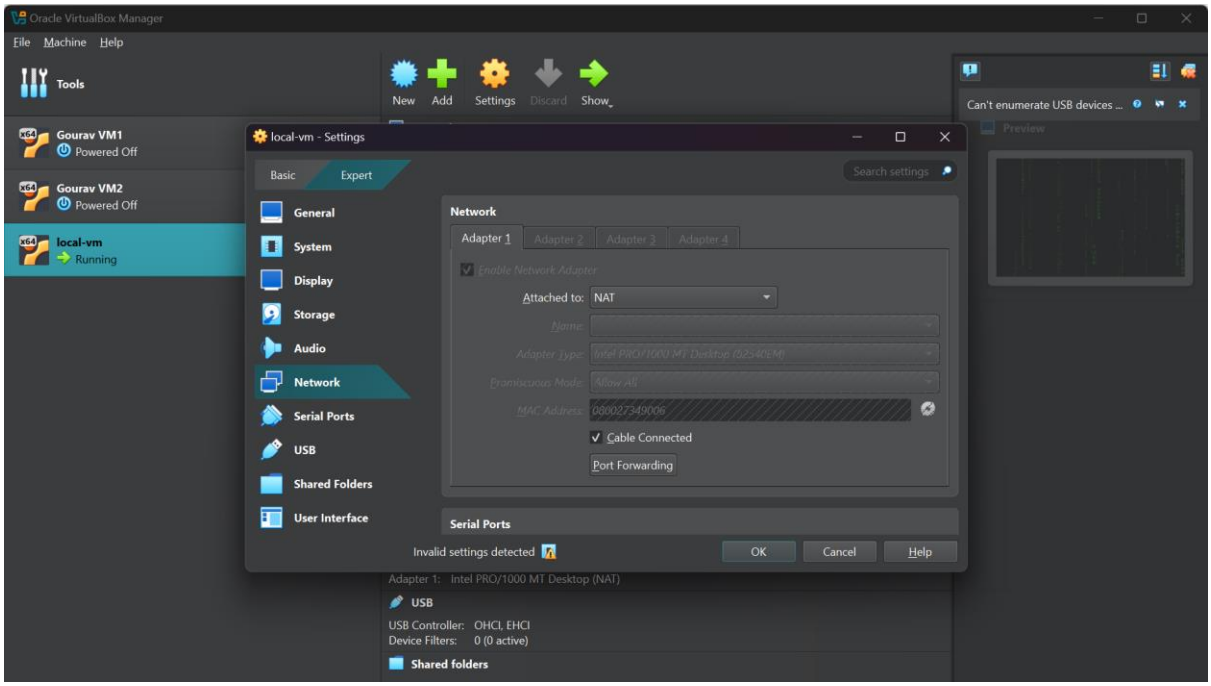


2.5 Configure VM Settings

1. Select the newly created VM and click **Settings**.

2. Go to **Network > Adapter 1**:
    - **Attached to:** NAT
    - Click **Advanced > Port Forwarding**.
3. Add the following rules:

- **SSH Rule:**
    - **Name:** SSH
    - **Protocol:** TCP
    - **Host IP:** 127.0.0.1
    - **Host Port:** 2222
    - **Guest Port:** 22
- **Web Application Rule:**
    - **Name:** WebApp
    - **Protocol:** TCP
    - **Host IP:** 127.0.0.1
    - **Host Port:** 8080
    - **Guest Port:** 80

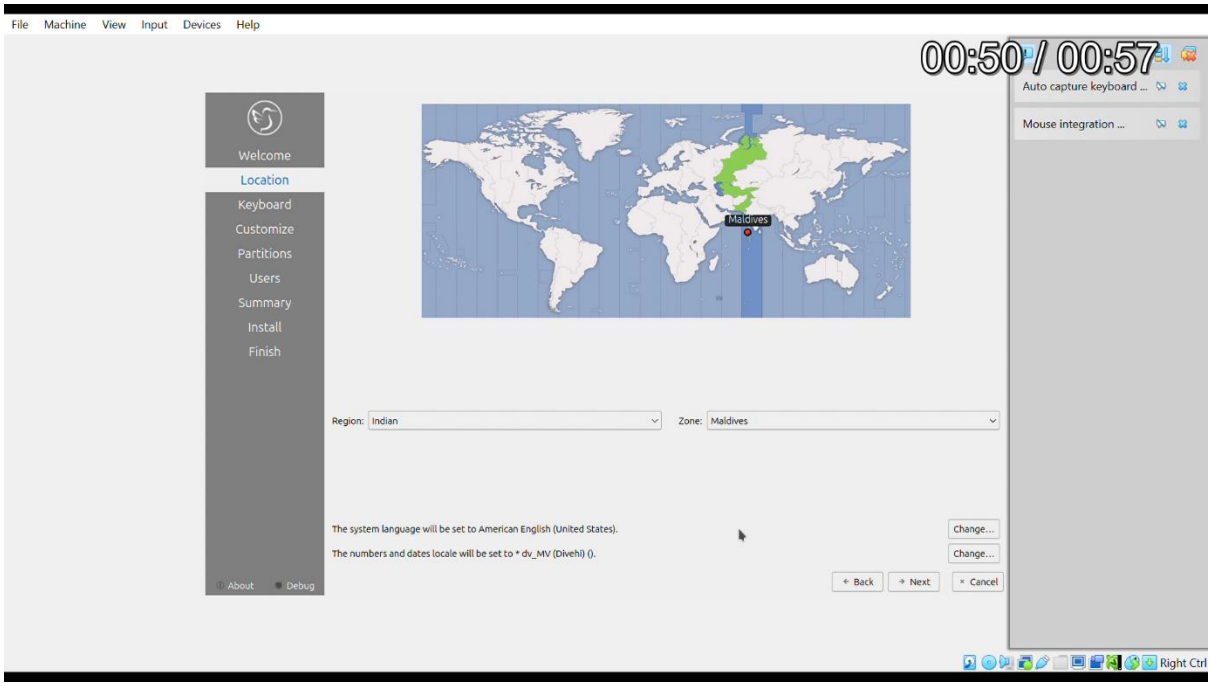


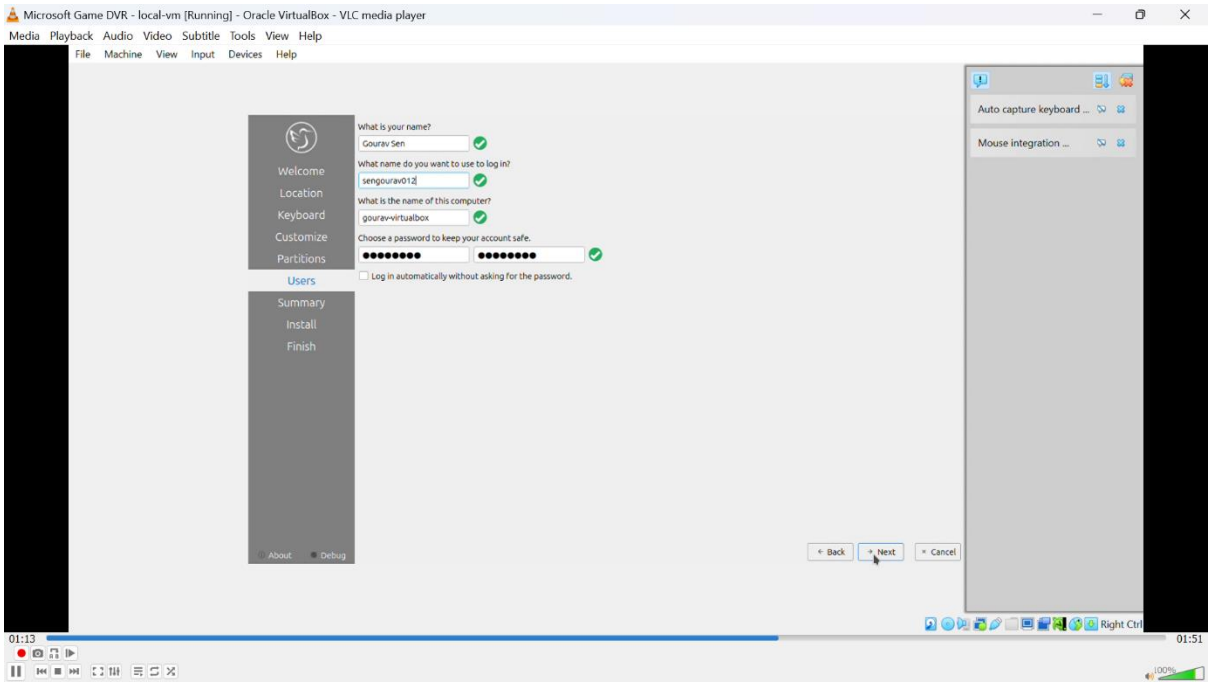
2.6 Install Lubuntu on the VM

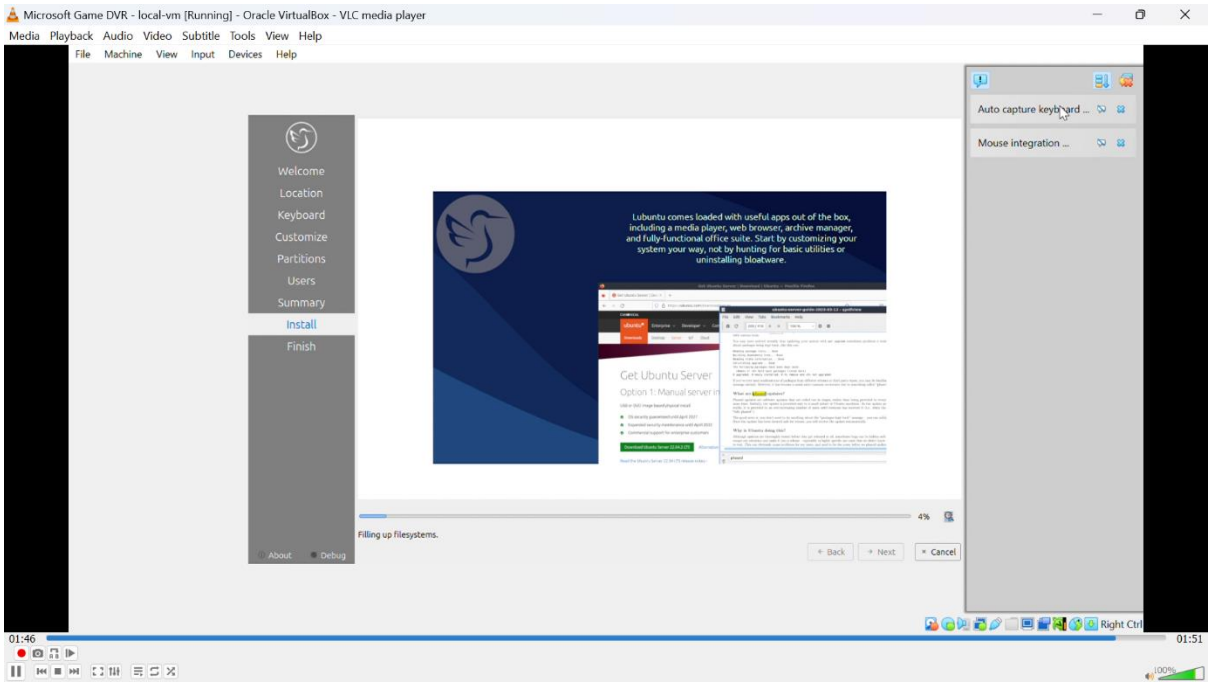
1. Start the VM and select the **Lubuntu ISO** to boot.
2. Choose **Install Lubuntu**.
3. Follow the on-screen instructions:
    - Select **Language, Timezone, and Keyboard Layout**.
    - Set username and password.
4. Once the installation is complete, **remove the ISO** and restart the VM.


File Machine View Input Devices Help
Auto capture keyboard ...
Mouse integration ...
lubuntu
Select Your Language
English (United States)
Internet Connection
Ethernet 1: Disconnected
Try Lubuntu
Install Lubuntu
local-vm [Running] - Oracle VirtualBox - Microsoft Game DVR
Right Ctrl

5.


File Machine View Input Devices Help
00:50 / 00:57
Auto capture keyboard ...
Mouse integration ...
Welcome
Location
Keyboard
Customize
Partitions
Users
Summary
Install
Finish
Maldives
Region: Indian
Zone: Maldives
The system language will be set to American English (United States).
The numbers and dates locale will be set to * dv_MV (Divehi) ().
Change...
Change...
About
Debug
Back
Next
Cancel
Right Ctrl





2.7 SSH into the VM

After the VM is up and running, SSH into the VM from your local machine.

First, find the local VM IP by running:

```
ip a
```

From the local machine, SSH into the VM:

```
ssh username@127.0.0.1 -p 2222
```

3. Docker Installation and Setup

3.1 Install Docker on the VM

Once you have SSH access to the VM, install Docker using the following steps.

3.1.1 Update the Package Index

sudo apt update

3.1.2 Install Required Dependencies

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

3.1.3 Add Docker's Official GPG Key

```
curl-fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

3.1.4 Add Docker Repository

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \

https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

3.1.5 Install Docker Engine

```
sudo apt update

sudo apt install -y docker-ce docker-ce-cli containerd.io
```

3.3 Add User to Docker Group

To run Docker without sudo, add your user to the Docker group:

```
sudo usermod -aG docker $USER

newgrp docker
```

4. Package Installation and Application Setup

containerize the web application

create app.py

```python
from flask import Flask, jsonify
import threading
import time
import random

app = Flask(__name__)

# Global variable to control the load
is_loading = False

def cpu_load():
    """Function to simulate continuous CPU load while web app is open."""
    global is_loading
    while is_loading:
        # Perform heavy computation
        [x**2 for x in range(10**7)]  # CPU intensive task
        time.sleep(0.1)

@app.route('/')
def home():
    return "Web App is running! Visit /load to increase CPU usage."

@app.route('/load')
def load():
    global is_loading
    if not is_loading:
        is_loading = True
        threading.Thread(target=cpu_load).start()
    return jsonify({"message": "CPU load started"})

@app.route('/stop')
def stop():
    global is_loading
    is_loading = False
    return jsonify({"message": "CPU load stopped"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

create Dockerfile with following content:

```
# Use lightweight Python image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the app files
COPY app.py /app

# Install dependencies
RUN pip install Flask

# Expose port 80
EXPOSE 80

# Run the app
CMD ["python", "app.py"]
```

Build the image:

```
docker build webapp
```

Create the Docker network

```
docker network create my-network
```

Run the Docker Container with Custom Names

Use the following command to run the web application with:

- **Container Name:** container
- **Network:** my-network
- **Port Mapping:** 80:80

```
docker run -d --name container --network my-network -p 80:80 webapp
```

✅ **4.4 Test the Application**

Access the application from the host machine:

```
curl http://127.0.0.1:8080
```

Step 8: Resource Monitoring Script
Objective:

- Create a script to monitor the CPU usage of the local VM.
- Trigger autoscaling by creating a GCP VM instance when the CPU usage exceeds **75%**.
- Perform live migration of the application to the GCP VM.
- Migrate the application back to the local VM when the CPU usage decreases below **40%**

Create the Resource Monitoring Script:
nano autoscale.sh

```
#!/bin/bash

# Variables
THRESHOLD=75            # Upper threshold to start the VM
MIN_THRESHOLD=30         # Lower threshold to stop the VM
INSTANCE_NAME="web-app-instance"
ZONE="asia-south1-a"
MACHINE_TYPE="e2-medium"
IMAGE_FAMILY="ubuntu-2204-lts"
IMAGE_PROJECT="ubuntu-os-cloud"

# Docker configuration
CONTAINER_NAME="container"
LOCAL_PORT=8080
DOCKER_PORT=80
DOCKER_IMAGE="webapp"

# --- Function to check if GCP VM is reachable ---
check_vm_ready() {
    echo "Checking VM status..."
    local retries=15  # Number of retries (30 seconds total)
    local delay=2     # Delay between retries

    for ((i=1; i<=retries; i++)); do
        if gcloud compute ssh "$INSTANCE_NAME" --zone="$ZONE" --strict-host-key-checking=no --command "echo 'VM is ready'" &>/dev/null; then
            echo "✅ VM is reachable!"
            return 0
        else
            echo "Waiting for VM to be reachable... Attempt $i/$retries"
            sleep "$delay"
        fi
    done

    echo "❌ Failed to connect to VM after $((retries * delay)) seconds."
    exit 1
}
```

```
# --- Check if the GCP VM exists ---
echo "Checking if GCP VM exists..."
if ! gcloud compute instances describe "$INSTANCE_NAME" --zone="$ZONE" &> /dev/null; then
    echo "GCP VM does not exist. Creating it..."

    # Create the GCP VM
    gcloud compute instances create "$INSTANCE_NAME" \
        --zone="$ZONE" \
        --machine-type="$MACHINE_TYPE" \
        --image-family="$IMAGE_FAMILY" \
        --image-project="$IMAGE_PROJECT" \
        --tags=http-server

    echo "Waiting for VM to start..."
    sleep 30

    # Ensure the VM is reachable before proceeding
    check_vm_ready

    # SSH into the VM and install Docker, pull image, and run container
    echo "Installing Docker and deploying the web app on the new VM..."

    gcloud compute ssh "$INSTANCE_NAME" --zone="$ZONE" --strict-host-key-checking=no --command "
        sudo apt update -y &&
        sudo apt install -y docker.io &&
        sudo systemctl start docker &&
        sudo systemctl enable docker &&
        sudo docker run -d --name $CONTAINER_NAME -p $DOCKER_PORT:$DOCKER_PORT $DOCKER_IMAGE
    "

    echo "✅ Web app deployed successfully on the new GCP VM!"
else
    echo "GCP VM already exists."
fi
```

```
# --- CPU Monitoring Section ---

# Get Docker container CPU usage
CONTAINER_CPU=$(docker stats --no-stream --format "{{.CPUPerc}}" "$CONTAINER_NAME" | sed 's/%//')

# Get VM CPU usage
VM_CPU=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')

# Combine CPU usage by adding container and VM CPU usage
TOTAL_CPU=$(echo "$CONTAINER_CPU + $VM_CPU" | bc -l)
echo "Container CPU: $CONTAINER_CPU%"
echo "VM CPU: $VM_CPU%"
echo "Total CPU Usage: $TOTAL_CPU%"

# --- Autoscaling Logic ---
if (( $(echo "$TOTAL_CPU > $THRESHOLD" | bc -l) )); then
    echo "⚠️ High CPU usage detected: $TOTAL_CPU%"

    if gcloud compute instances describe "$INSTANCE_NAME" --zone="$ZONE" &> /dev/null; then
        echo "Starting existing GCP VM: $INSTANCE_NAME"
        gcloud compute instances start "$INSTANCE_NAME" --zone="$ZONE"

        # Wait until the VM is ready
        check_vm_ready

    else
        echo "Creating and starting a new GCP VM: $INSTANCE_NAME"
        gcloud compute instances create "$INSTANCE_NAME" \
            --zone="$ZONE" \
            --machine-type="$MACHINE_TYPE" \
            --image-family="$IMAGE_FAMILY" \
            --image-project="$IMAGE_PROJECT" \
            --tags=http-server

        echo "Installing Docker and deploying web app..."
```

```
        echo "Installing Docker and deploying web app..."

        gcloud compute ssh "$INSTANCE_NAME" --zone="$ZONE" --strict-host-key-checking=no --command "
            sudo apt update -y &&
            sudo apt install -y docker.io &&
            sudo systemctl start docker &&
            sudo systemctl enable docker &&
            sudo docker run -d --name $CONTAINER_NAME -p $DOCKER_PORT:$DOCKER_PORT $DOCKER_IMAGE
        "

        echo "✅ Web app deployed successfully!"
    fi

elif (( $(echo "$TOTAL_CPU < $MIN_THRESHOLD" | bc -l) )); then
    echo "✅ Low CPU usage detected: $TOTAL_CPU%"
    echo "Stopping GCP VM: $INSTANCE_NAME"
    gcloud compute instances stop "$INSTANCE_NAME" --zone="$ZONE"

else
    echo "✅ CPU usage is within normal range: $TOTAL_CPU%"
fi
```

Make the Script Executable:

```
chmod +x autoscale.sh
```

Step 9: Google Cloud SDK Installation and Service Account Setup

Objective:

- Install the Google Cloud SDK to interact with GCP.
- Authenticate with GCP using a service account.
- Configure the SDK with the Asia-South1 (Mumbai) region.
- Verify GCP connectivity.

Add the Cloud SDK distribution URI as a package source

```
echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] http://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
```

Import the Google Cloud public key

```
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/cloud.google.gpg
```

Update and install Google Cloud SDK

```
sudo apt update && sudo apt install -y google-cloud-sdk
```

Verify the installation by running:

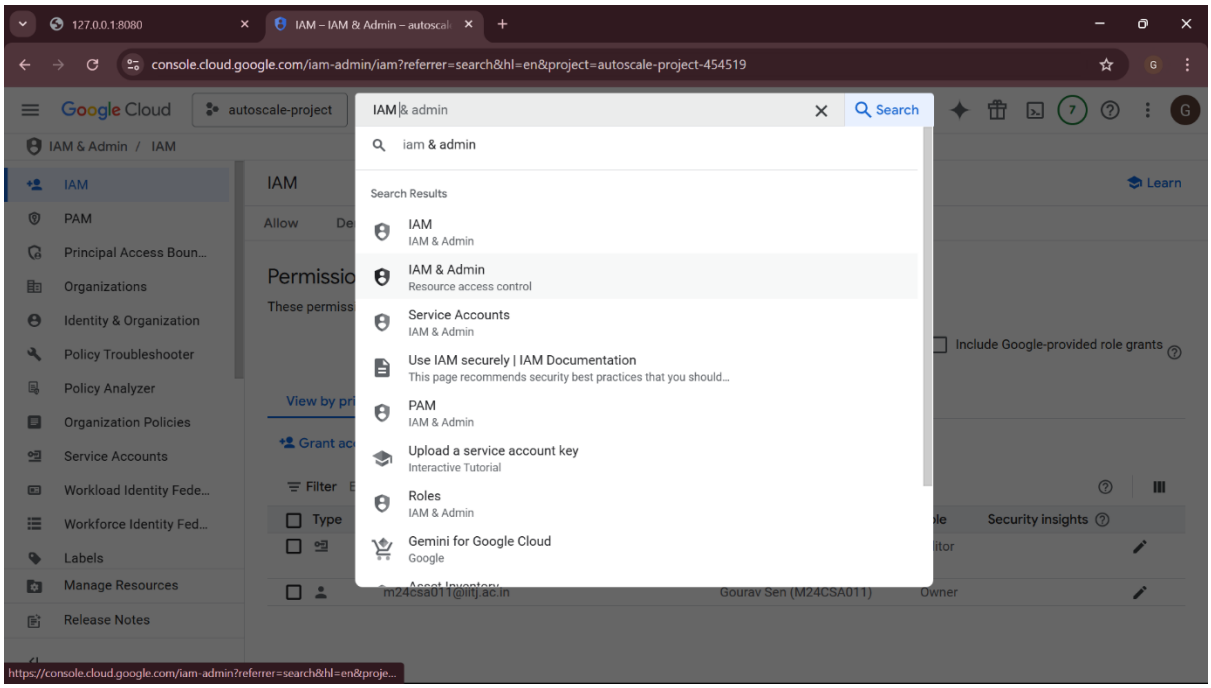
```
gloud version
```

If this appears you are good to go.

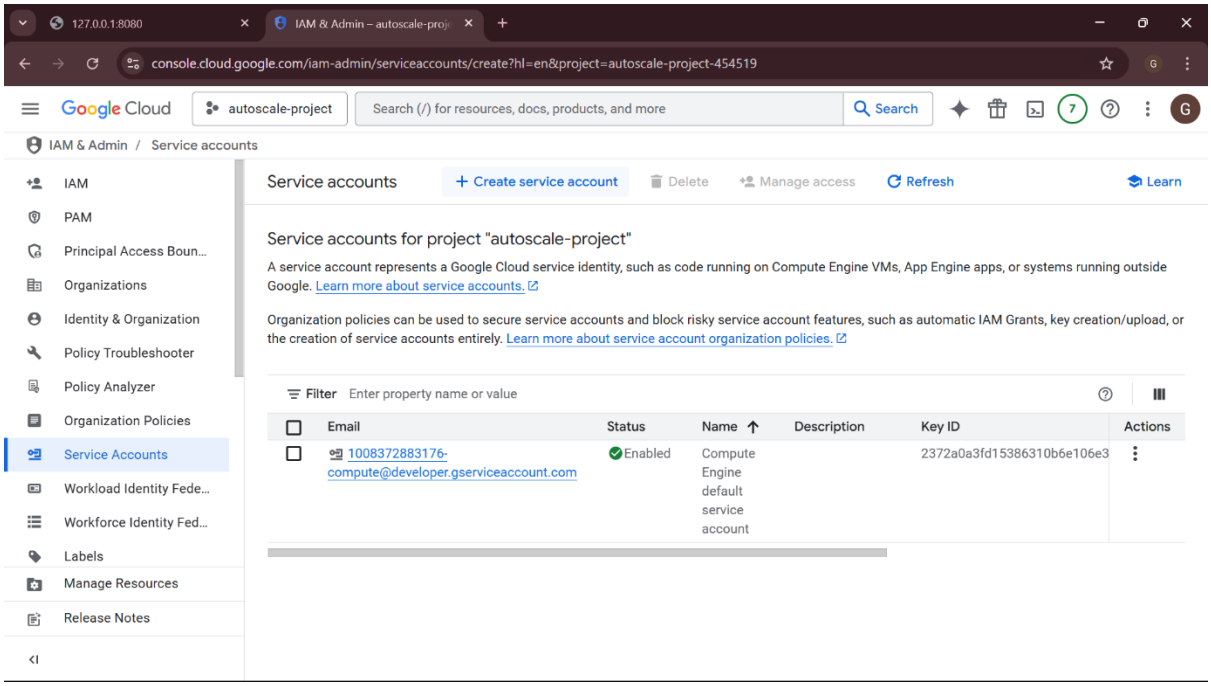
```
Google Cloud SDK 465.0.0
bq 2.0.88
core 2024.03.01
gsutil 5.30
```

Create Service Account :

Go to IAM and Admin:

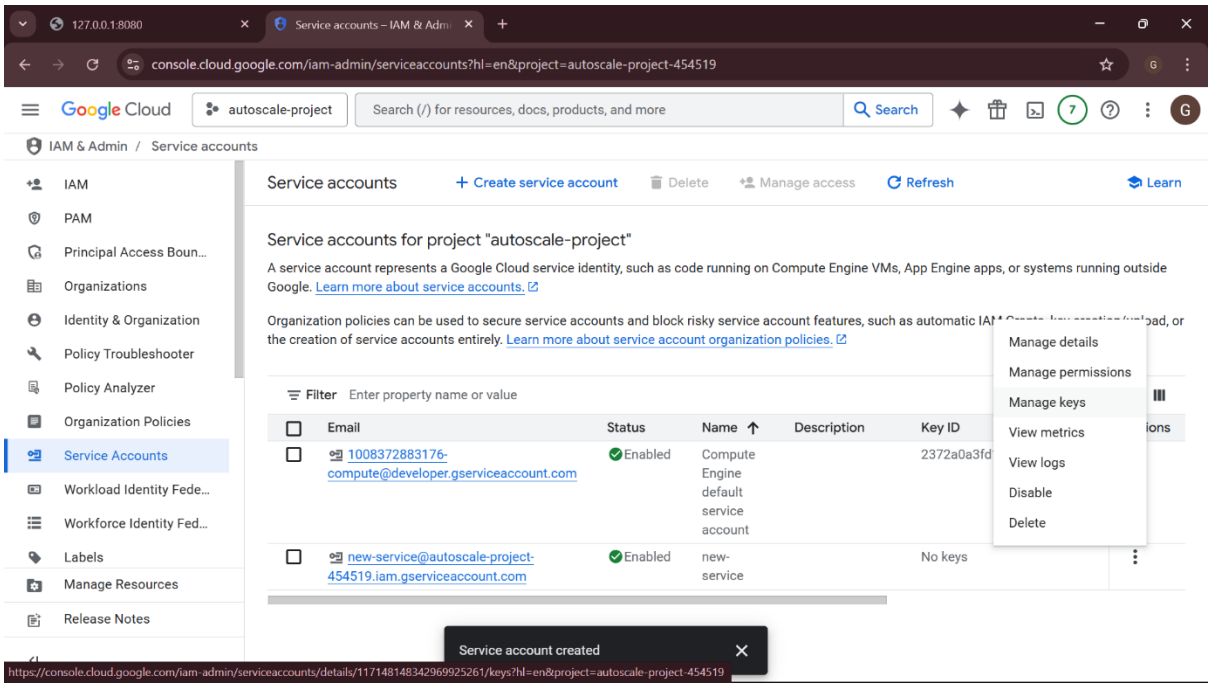


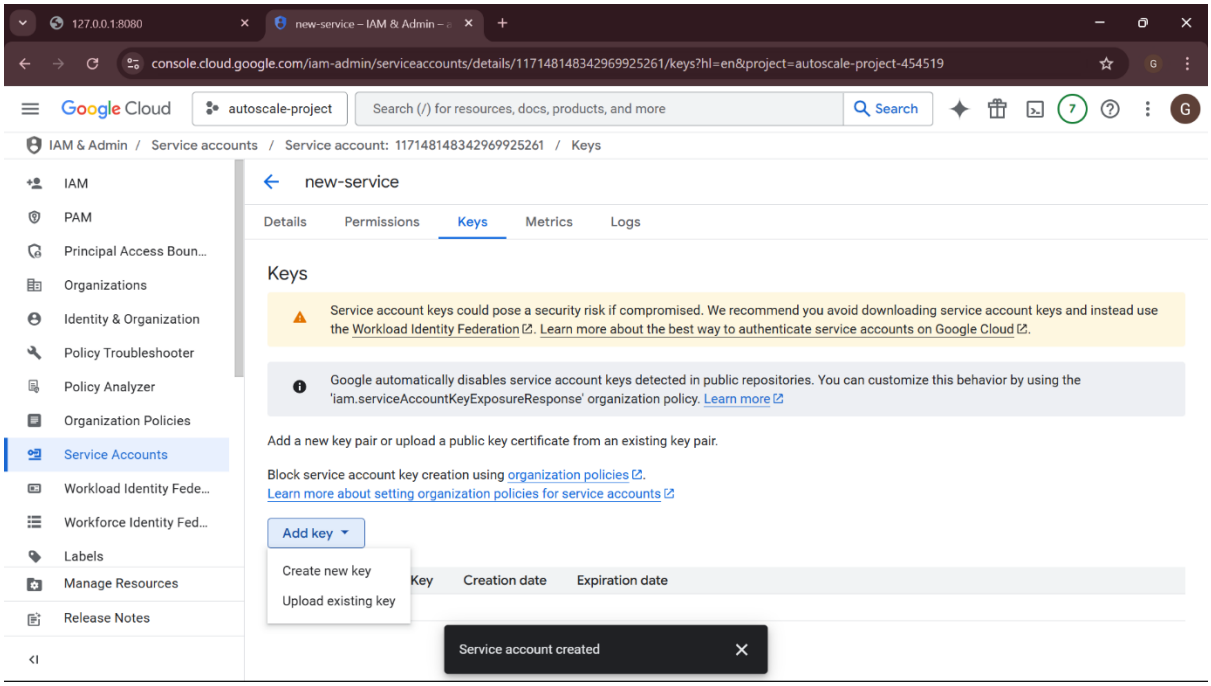
Go to Service account:

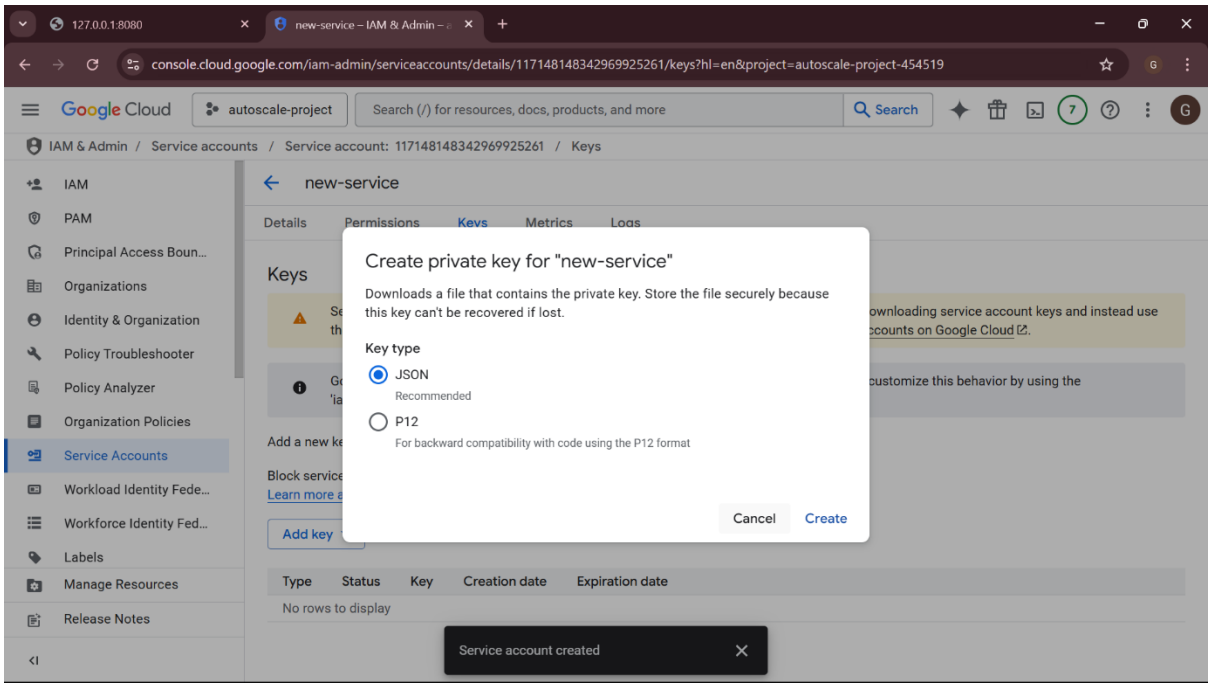


Create service account here.

After creating service account we need to create a key as well, this will be used by our VM to authenticate with the GCP –







This will download the json file with authentication key.

Authenticate with the service account using the key:

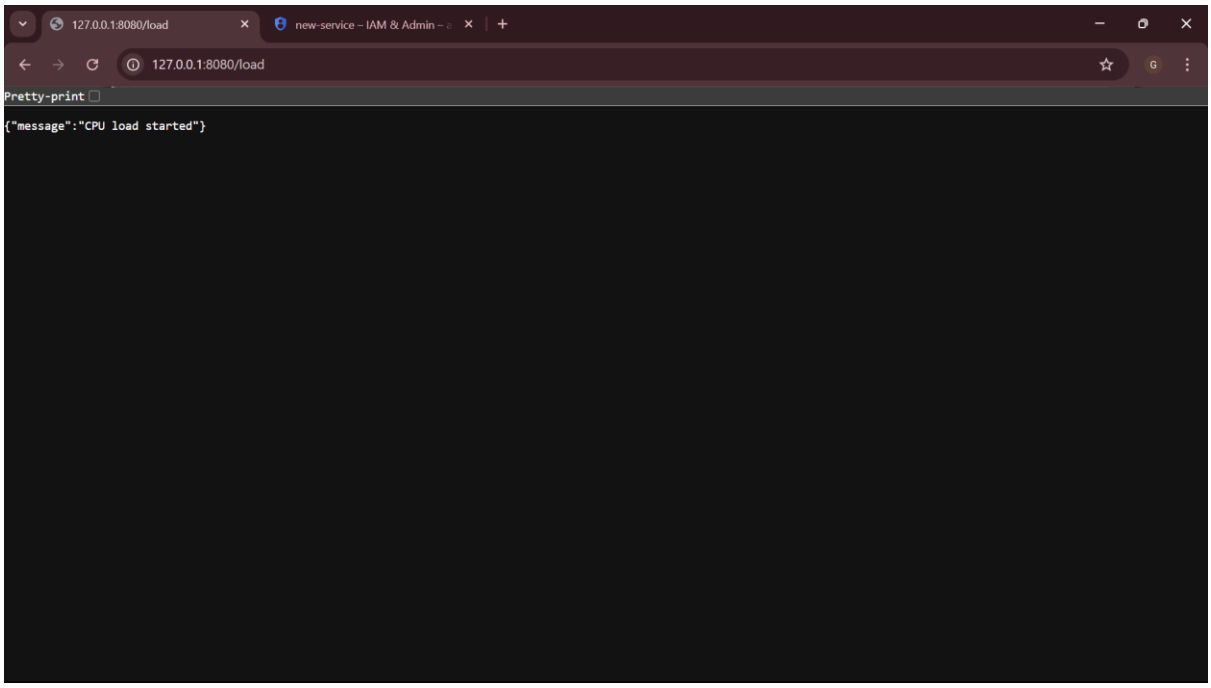
```
gcloud auth activate-service-account --key-file ~/gcp-key.json
```

Final Results and Output:

Now after doing all the setups we can check the autoscaling part:

Lets give load to site:

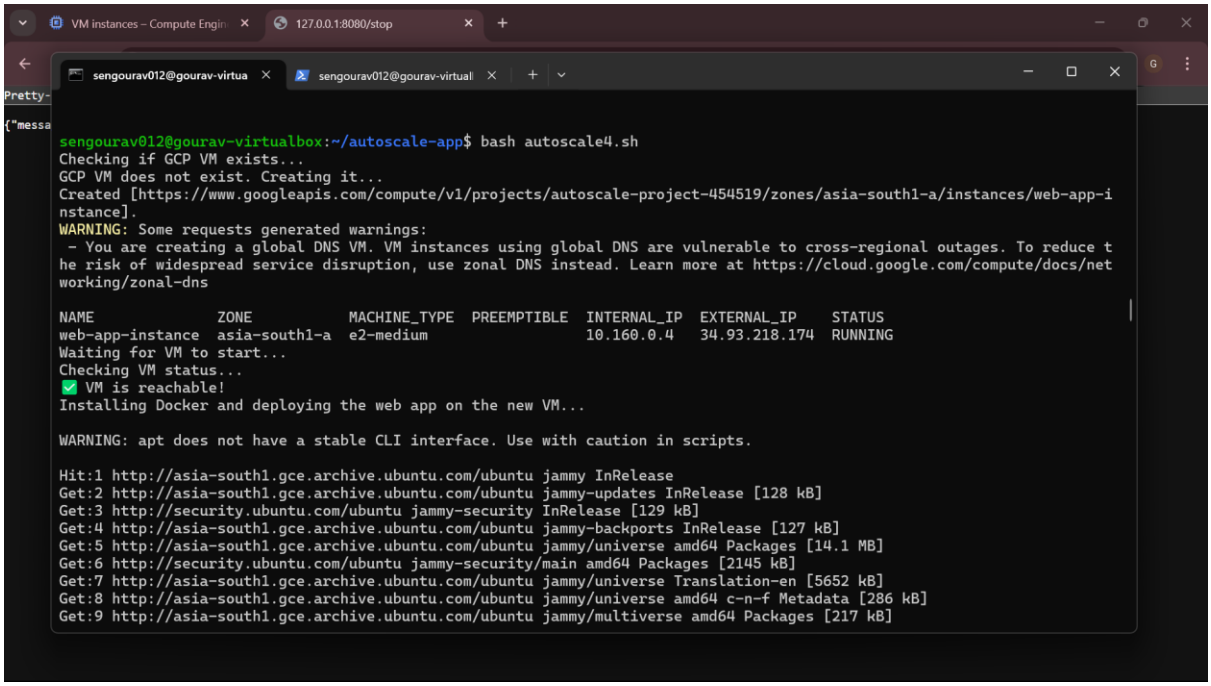
Run http://127.0.0.1:8080/load in the browser:



Run the monitoring script:

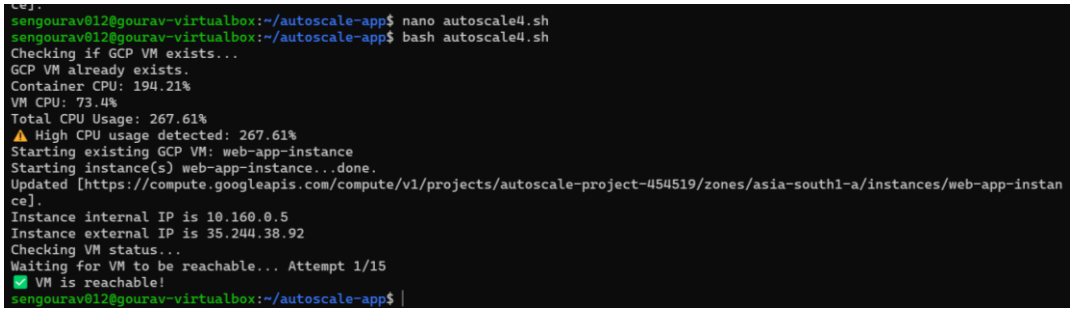
```
sengourav012@gourav-virtualbox:~/autoscale-app$ bash autoscale4.sh
```

Result:



We can see that in the case of load (above 75%) , a new vm instance is being created in google cloud platform.

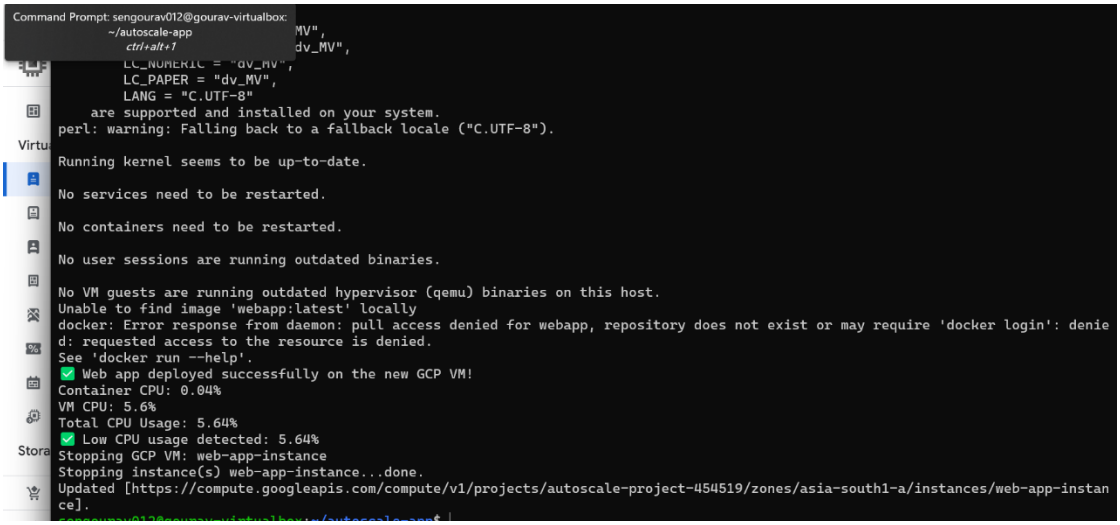
Run monitoring script:



We can see that the load is being transferred to GCP VM instance.

Lets decrease the load.

Run http://127.0.0.1:8080/stop in browser:
Output:



Clearly the the VM instance is stopped, when the load is below 40% (min threshold).

Conclusion:

The project successfully demonstrated **auto-scaling and live migration** of a web application from a **local VM to GCP** based on CPU usage thresholds. The **monitoring script** effectively triggered the scaling event when CPU usage exceeded **75%**, deploying the application to a GCP VM. Upon load reduction, the application automatically migrated back to the local VM. This solution ensures **efficient resource utilization**, seamless scaling, and reduces infrastructure costs by dynamically managing cloud resources.