Virtualization and Cloud Computing
Assignment 3

Name: Rishabh Acharya
Roll Number: B22CS090

Demonstration Link:
https://youtu.be/jHu_mUyrJiE

GitHub Link:
https://github.com/rishz09/virtualization-and-cloud-computing/tree/main/assignment3

NOTE: It is recommended to read about the code below, before watching the demonstration if you want to have a better understanding of how the code works.

Local Virtual Machine

- In VirtualBox, I have created a local linux VM by allocating 20GB storage, 8GB VRAM and 4 cores.
- This local VM will auto-scale to GCP when CPU usage goes above 75% consistently for 10 seconds.
- The GCP migrated tasks will also migrate back to local VM when CPU usage stays below 60% consistently for 10 seconds.

Resource Monitoring Tool - psutil

psutil is a Python library that provides an interface for retrieving information on system utilization (CPU, memory, disks, network, sensors) and managing running processes.

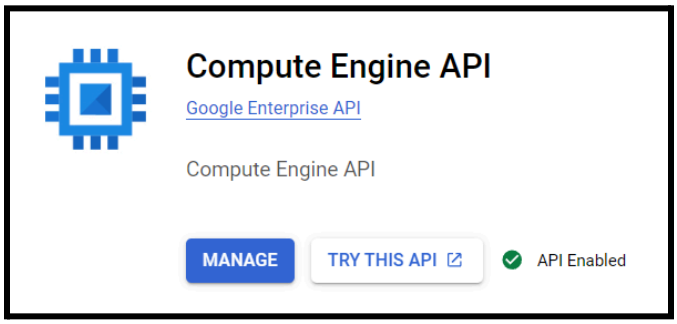
Example of simple resource monitoring using psutil:

```
import psutil
import time

while True:
    cpu_usage = psutil.cpu_percent(interval=1)
    mem_usage = psutil.virtual_memory().percent

    print(f"CPU Usage: {cpu_usage}% \t Memory Usage: {mem_usage}%")

    time.sleep(2)
```

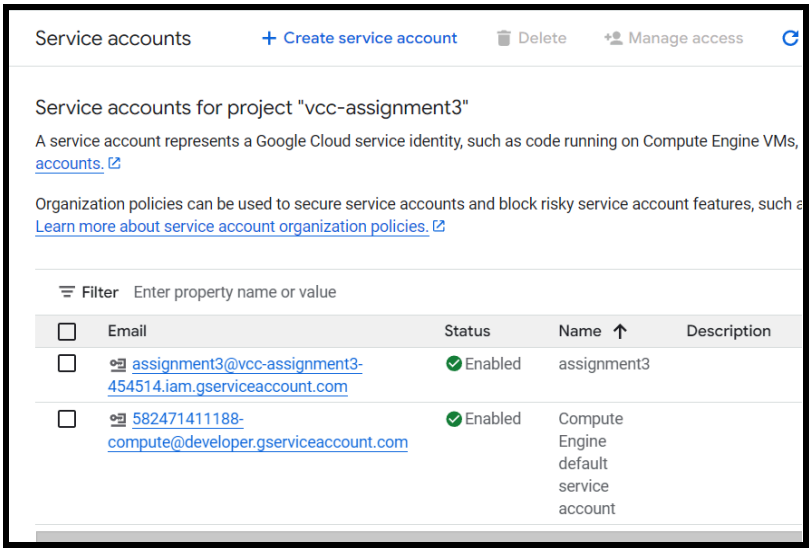
```
CPU Usage: 0.8%          Memory Usage: 31.3%
CPU Usage: 7.5%          Memory Usage: 31.3%
CPU Usage: 0.3%          Memory Usage: 31.3%
CPU Usage: 1.5%          Memory Usage: 31.3%
CPU Usage: 6.5%          Memory Usage: 31.3%
CPU Usage: 0.3%          Memory Usage: 31.3%
CPU Usage: 0.5%          Memory Usage: 31.3%
CPU Usage: 0.5%          Memory Usage: 31.3%
```

Steps to enable GCP to autoscale the local VM

1. Enable Compute Engine API.



2. Create a Service Account for Automation
   - Navigate to IAM & Admin > Service Accounts.
   - Click on Create Service Account:
   - Give it a name (e.g: assignment3).
   - Assign the following roles to service account so that migration can take place:
     - ➔ Compute Admin.
     - ➔ Service Account User.



3. Generate and download a JSON Key
   - After creating the service account:
   - Go to the Keys section.
   - Click Add Key > Create New Key.
   - Choose JSON format.
   - Download the JSON key file (e.g: autoscaler-key.json).

4. Install and configure gcloud CLI on local VM
   - This is needed in order to give commands to GCP from local VM.
   - Steps to install gcloud CLI:
     sudo apt-get update
     sudo apt-get install -y apt-transport-https ca-certificates gnupg
     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
     sudo apt-get update && sudo apt-get install -y google-cloud-sdk

   - Authenticate Local VM with Service Account JSON Key. This enables the local VM to communicate with the given service account.
     gcloud auth activate-service-account --key-file=~/autoscaler-key.json

   - Set the project ID. This specifies the project where VM instances are to be created.
     gcloud config set project YOUR_PROJECT_ID

   - gcloud compute instances list --project=PROJECT_ID

The above command should now show the VM instances associated with this project.

```
NAME     ZONE           MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP
 STATUS
vm-test  asia-south2-a  n1-standard-1               10.190.0.8   34.126.213.170
 RUNNING
```

Now, GCP has been set up properly and we need a simple application to demonstrate the migration of tasks to GCP, when CPU usage of local VM exceeds threshold.

Working of the application

- The python script simulates a local server which has 4 resource intensive tasks running parallelly.
- It dynamically manages CPU-intensive tasks by monitoring local CPU usage and migrating workloads to a GCP VM when necessary.
- It starts by loading threshold values and timing configurations from a config file.
- Locally, it runs four CPU-intensive tasks as separate processes.
- If the CPU usage exceeds a specified high threshold for a continuous period, two tasks are migrated to a newly created GCP VM to reduce local load.
- The script automates the creation of the VM, transfers the **gcp_tasks.py** script using **SCP**, and starts the tasks remotely using **nohup**. It also streams logs from the remote VM back to the local console.
- When CPU usage stabilizes below a low threshold, the tasks are brought back to the local machine, and the remote tasks are stopped.
- Note that the VM is not deleted once the 2 tasks are migrated back to the local VM again. The tasks are just stopped in the GCP VM. This is done so that if tasks are required to be migrated to GCP again, we will save time by not creating a new VM again.
- GCP VM is deleted only when the user stops executing the whole python script. It also kills the local processes.
- The entire system is designed to balance load dynamically between the local machine and the cloud based on real-time CPU usage.

Breakdown of the Code

Attributes / configurations:

- **THRESHOLD_HIGH, THRESHOLD_LOW:** CPU usage thresholds to trigger migration (high to migrate, low to return).
- **CHECK_INTERVAL:** How often (in seconds) the CPU usage is checked.
- **HIGH_CPU_DURATION, LOW_CPU_DURATION:** Time durations to confirm sustained high/low CPU usage before acting.
- **PROJECT_ID, ZONE, MACHINE_TYPE, etc:** GCP configuration for VM creation.
- **vm_name:** Stores the name of the current GCP VM.
- **migrated:** Flag indicating if tasks are currently migrated to the GCP VM.
- **vm_log_thread_stop_event:** Signals the log streaming thread to stop.

Local Task Management:

- **cpu_intensive_task(task_name):** Simulates an infinite CPU-intensive loop for demonstration.
- **start_tasks(task_names, processes):** Starts local CPU-intensive tasks as separate processes and tracks them.
- **stop_tasks(task_names, processes):** Stops and cleans up local processes by terminating them.

GCP VM Management / Task Migration

- **create_gcp_vm_if_needed():** Creates a GCP VM (if one doesn't exist), waits for it to boot, and copies gcp_tasks.py to it using **SCP**. SCP is used in this script to **securely transfer files** from the local machine to the remote GCP VM instance.
- **start_tasks_on_gcp_vm():** SSH into the VM and runs gcp_tasks.py in the background using **nohup** (for persistent execution).
- **stop_tasks_on_gcp_vm():** SSH into the VM and kills the gcp_tasks.py processes.
- **delete_gcp_vm():** Deletes the GCP VM and stops the log streaming thread.

Logging

stream_gcp_vm_logs(vm_name): Continuously streams the **gcp_tasks.py** log output from the GCP VM to the local console in a separate thread.

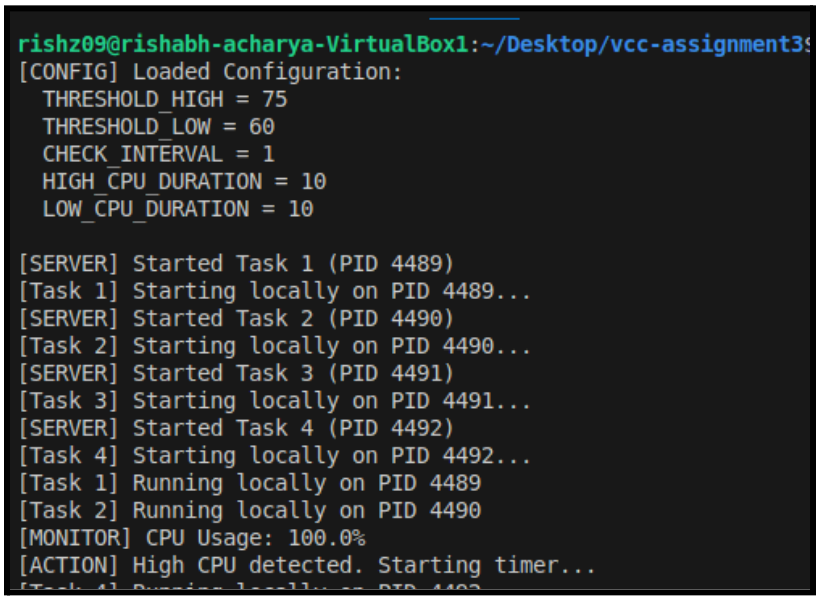
Monitoring / Management of Migration

monitor_and_manage(): Does the following tasks:

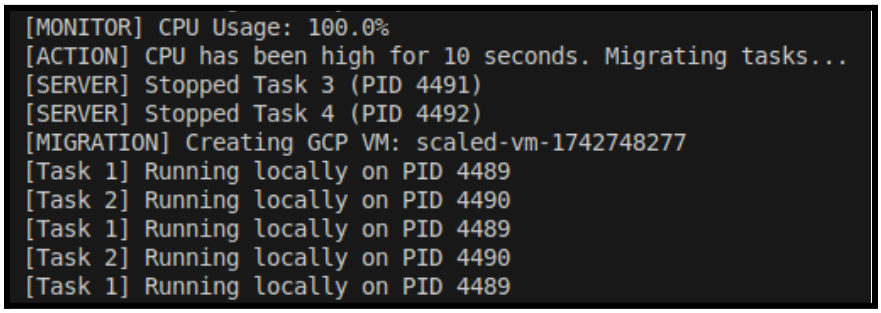
- Monitors local CPU usage.
- Migrates tasks to GCP if CPU is high for a certain duration.
- Brings tasks back if CPU stabilizes.
- Handles user interruptions and triggers cleanup of local processes and GCP VM.

Some screenshots to show working of code

- Starting of code

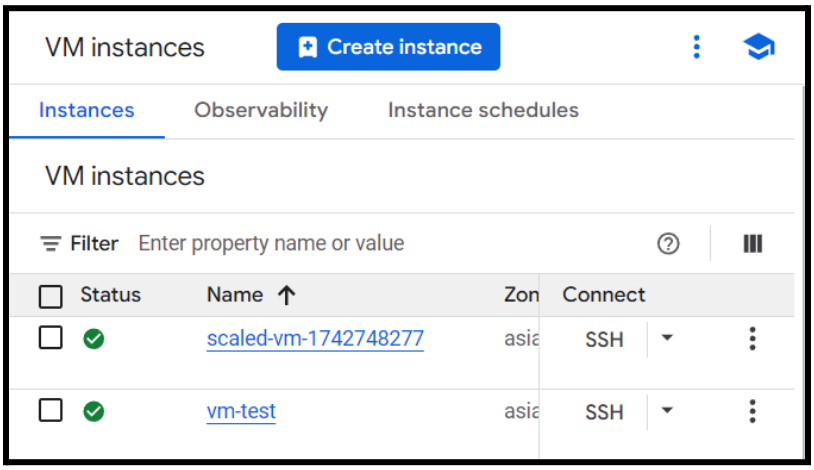


- CPU usage has been high for 10 seconds and 2 tasks are to be migrated to GCP VM. Notice that only 2 tasks are currently running on local VM now.



- GCP VM has been created and now we wait for 30 seconds for GCP VM to boot properly.

```
Created [https://www.googleapis.com/compute/v1/projects/vcc-assignment3-454514/zones/asia-south
2-a/instances/scaled-vm-1742748277].
WARNING: Some requests generated warnings:
 - You are creating a global DNS VM. VM instances using global DNS are vulnerable to cross-regi
onal outages. To reduce the risk of widespread service disruption, use zonal DNS instead. Learn
 more at https://cloud.google.com/compute/docs/networking/zonal-dns

NAME                ZONE           MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP    ST
ATUS
scaled-vm-1742748277  asia-south2-a  n1-standard-1              10.190.0.14  34.131.87.209  RU
NNING
[MIGRATION] VM scaled-vm-1742748277 created successfully!
[MIGRATION] Waiting 30 seconds for VM to boot...
```



(vm-test is not related to this code)

- After booting, a script is copied from local VM to GCP VM using SCP, which runs tasks 3 and 4.

```
Warning: Permanently added 'compute.7362983629733856406' (ED25519) to the list of known hosts.
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
gcp_tasks.py                                  100%  581    10.9KB/s   00:00
[MIGRATION] gcp_tasks.py copied to VM scaled-vm-1742748277.
[LOG] Streaming logs from scaled-vm-1742748277...
[MIGRATION] Starting tasks on GCP VM scaled-vm-1742748277...
```

- Tasks 3 and 4 are now running on GCP VM. Also, local CPU usage is low, so a timer is kept running.

```
[MIGRATION] Started gcp_tasks.py on VM scaled-vm-1742748277.
[GCP VM LOG] [Task 3] Starting on GCP!
[GCP VM LOG] [Task 4] Starting on GCP!
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[MONITOR] CPU Usage: 39.7%
[ACTION] CPU low. Starting timer to bring tasks back...
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[GCP VM LOG] [Task 3] Running on GCP
[GCP VM LOG] [Task 4] Running on GCP
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[MONITOR] CPU Usage: 40.1%
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[MONITOR] CPU Usage: 48.7%
[GCP VM LOG] [Task 3] Running on GCP
[Task 1] Running locally on PID 4489
[GCP VM LOG] [Task 4] Running on GCP
```

- When CPU has been at low usage consistently for required time (10 seconds here), tasks 3 and 4 are migrated back to local VM and stopped running on GCP VM. Note that PIDs of task 3 and task 4 are different from the first screenshot, which indicates that they have newly started.

```
[ACTION] CPU stable. Bringing tasks back locally...
[MIGRATION] Stopping tasks on GCP VM scaled-vm-1742748277...
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[GCP VM LOG] [Task 3] Running on GCP
[GCP VM LOG] [Task 4] Running on GCP
[Task 2] Running locally on PID 4490
[Task 1] Running locally on PID 4489
[MIGRATION] Stopped tasks on GCP VM scaled-vm-1742748277.
[SERVER] Started Task 3 (PID 4970)
[Task 3] Starting locally on PID 4970...
[Task 4] Starting locally on PID 4971...
[SERVER] Started Task 4 (PID 4971)
[Task 2] Running locally on PID 4490
[Task 1] Running locally on PID 4489
[Task 3] Running locally on PID 4970
```

- From next time, when migration is again required, a new GCP VM is not created, rather the old one is used again.

```
[MONITOR] CPU Usage: 100.0%
[ACTION] CPU has been high for 10 seconds. Migrating tasks...
[SERVER] Stopped Task 3 (PID 4970)
[SERVER] Stopped Task 4 (PID 4971)
[MIGRATION] GCP VM scaled-vm-1742748277 already exists. Skipping creation.
[MIGRATION] Starting tasks on GCP VM scaled-vm-1742748277...
[Task 2] Running locally on PID 4490
[Task 1] Running locally on PID 4489
[Task 2] Running locally on PID 4490
[Task 1] Running locally on PID 4489
[GCP VM LOG] [Task 3] Starting on GCP!
[GCP VM LOG] [Task 4] Starting on GCP!
[MIGRATION] Started gcp_tasks.py on VM scaled-vm-1742748277.
```

- This cycle keeps on continuing till the execution of the script is interrupted. That's when the local processes and GCP VM are deleted.

```
^C[MONITOR] Stopped by user. Cleaning up...
[SERVER] Stopped Task 1 (PID 4489)
[SERVER] Stopped Task 2 (PID 4490)
[SERVER] Stopped Task 3 (PID 5184)
[SERVER] Stopped Task 4 (PID 5186)
[CLEANUP] Deleting VM scaled-vm-1742748277...
Deleted [https://www.googleapis.com/compute/v1/projects/vcc-assignment3-454514/zones/asia-south
2-a/instances/scaled-vm-1742748277].
[CLEANUP] VM scaled-vm-1742748277 deleted.
```

Diagram Illustrating the Flow