Assignment - 3

Create a Local VM and Auto-Scale It to GCP When Resource Usage Exceeds 75%

Configuration

Submitted By
Anuj Chincholikar
B22ES018

March 23, 2025

Contents

1 Introduction

This report provides a detailed implementation of Assignment 3, focusing on creating a local virtual machine (VM) using VirtualBox, monitoring its resource usage with Prometheus and Node Exporter, and auto-scaling to Google Cloud Platform (GCP) when CPU usage exceeds 75%. A sample application is deployed to demonstrate the process, and the setup is rigorously tested to ensure functionality. The deliverables include this comprehensive document report, an architecture design, a source code repository, and a video demo.

The tools used in this project are:

- **VirtualBox**: For creating and managing the local VM.
- **Ubuntu 22.04 LTS**: As the operating system for the VM.
- **Prometheus and Node Exporter**: For monitoring system resources.
- **GCP Compute Engine**: For auto-scaling and cloud deployment.

The objective is to simulate a real-world scenario where a local system dynamically scales to a public cloud to handle increased workloads, ensuring high availability and resource efficiency. This implementation highlights the integration of open-source monitoring tools with cloud platforms, offering a cost-effective solution for resource management. Challenges such as network configuration, monitoring accuracy, and cloud authentication are addressed, providing practical insights into cloud computing and system administration.

Note: This project emphasizes automation and scalability, key concepts in modern cloud infrastructure, making it a valuable learning experience for system administration and DevOps practices.

2 Step-by-Step Implementation

2.1 Creating a Local VM with Ubuntu in VirtualBox

2.1.1 Prerequisites

- Host machine: Windows 11, 16GB RAM, Intel i7 processor.
- Internet connection for downloading VirtualBox and Ubuntu ISO.
- Virtualization support (VT-x/AMD-V) enabled in BIOS.

2.1.2 Install VirtualBox

- Downloaded VirtualBox 7.0 from `https://www.virtualbox.org/`.
- Installed the VirtualBox Extension Pack to enable USB support, shared folders, and enhanced display features.
- Verified the installation by launching VirtualBox and ensuring the interface loaded correctly.

Challenges Faced:

- Encountered a compatibility issue with Hyper-V on Windows, resolved by disabling Hyper-V in the Windows Features settings.
- Ensured VT-x/AMD-V virtualization was enabled in the BIOS to support 64-bit VMs.

Troubleshooting Tips:

- If VirtualBox fails to start, check for conflicting virtualization software (e.g., Hyper-V, VMware).

- Ensure the host machine has sufficient resources (at least 8GB RAM recommended).

2.1.3 Download Ubuntu ISO

- Downloaded Ubuntu 22.04 LTS ISO (approximately 3.5GB) from `https://ubuntu.com/download`.
- Verified the ISO checksum using the command: `sha256sum ubuntu-22.04.3-desktop-amd64.iso`.

Alternative Approach:

- Considered using Ubuntu 20.04 LTS for better compatibility with older systems but opted for 22.04 for the latest features and security updates.

2.1.4 Create a New Virtual Machine

- In VirtualBox, clicked *New* and configured:
- Name: `Ubuntu-VM`, Type: Linux, Version: Ubuntu (64-bit).
- Memory: Allocated 4GB (4096 MB) for smooth performance.
- Hard Disk: Created a VDI, dynamically allocated, size 20GB.

Additional Configuration:

- Enabled nested virtualization to support potential containerization (e.g., Docker) in future experiments.
- Set the VM to boot in EFI mode for better compatibility with modern systems.

2.1.5 Configure VM Settings

- General ¿ Advanced: Set Shared Clipboard and Drag'n'Drop to "Bidirectional".
- System ¿ Processor: Allocated 2 CPUs.
- Display ¿ Screen: Set Video Memory to 128MB, enabled 3D acceleration.
- Storage: Attached the Ubuntu ISO to the IDE Controller.
- Network: Set to "Bridged Adapter" for internet access.

Network Considerations:

- Initially considered NAT but switched to Bridged Adapter to allow external access to services.
- Assigned a static IP to the VM for consistent access.

2.1.6 Install Ubuntu on the VM

- Started the VM and selected "Install Ubuntu".
- Configured: Language (English), Keyboard (Default), Normal installation with updates, erased disk, set username (`sumit`) and password (`password123`).
- Installation took approximately 15 minutes, followed by a restart and ISO ejection.

Post-Installation Steps:

- Updated the system: `sudo apt update && sudo apt upgrade -y`.
- Installed essential tools: `sudo apt install -y build-essential`.

2.1.7 Install VirtualBox Guest Additions

- Inserted Guest Additions CD image via Devices menu.
- Ran: `sudo sh /media/sumit/VBox_GAs_*/VBoxLinuxAdditions.run`.
- Rebooted the VM: `sudo reboot`.

Benefits:

- Enabled full-screen resolution, shared folders, and improved graphics performance.
- Tested shared folders by mounting a directory from the host to the VM.

2.2 Implementing Resource Monitoring

2.2.1 Prerequisites

- Ubuntu VM up and running with internet access.
- Basic knowledge of system monitoring and Prometheus.

2.2.2 Install Monitoring Tools (Prometheus and Node Exporter)

- Updated package list: `sudo apt update`.
- Installed prerequisites: `sudo apt install -y curl`.

Install Node Exporter:

- Downloaded Node Exporter version 1.6.1 from: https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux-amd64.tar.gz.
- Extracted the tarball: `tar -xvf node_exporter-1.6.1.linux-amd64.tar.gz`.
- Moved the binary: `sudo mv node_exporter-1.6.1.linux-amd64/node_exporter /usr/local/bin/`.
- Configured as a systemd service:

```
sudo useradd -rs /bin/false node_exporter
sudo bash -c 'cat <<EOF > /etc/systemd/system/node_exporter.service
[Unit]
Description=Node Exporter
After=network.target
[Service]
User=node_exporter
ExecStart=/usr/local/bin/node_exporter
Restart=always
[Install]
WantedBy=multi-user.target
EOF'
```

- Enabled and started the service: `sudo systemctl enable node_exporter && sudo systemctl start node_exporter`.

Additional Configuration:

- Configured Node Exporter to collect disk I/O and memory usage metrics.
- Tested the service: `systemctl status node_exporter`.

Install Prometheus:

- Downloaded Prometheus version 2.45.0 from: https://github.com/prometheus/prometheus/releases/download/v2.45.0/prometheus-2.45.0.linux-amd64.tar.gz.
- Extracted the tarball: `tar -xvf prometheus-2.45.0.linux-amd64.tar.gz`.
- Moved the binary: `sudo mv prometheus-2.45.0.linux-amd64/prometheus /usr/local/bin/`.
- Created a configuration file:
  `sudo mkdir /etc/prometheus &&`
  `sudo nano /etc/prometheus/prometheus.yml`.

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'node'
    static_configs:
      - targets: ['localhost:9100']
```

- Configured as a systemd service:

```
sudo useradd -rs /bin/false prometheus
sudo bash -c 'cat <<EOF > /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus
After=network.target
[Service]
User=prometheus
ExecStart=/usr/local/bin/prometheus
   --config.file=/etc/prometheus/prometheus.yml
Restart=always
[Install]
WantedBy=multi-user.target
EOF'
```

- Enabled and started the service: `sudo systemctl enable prometheus && sudo systemctl start prometheus`.

Challenges Faced:

- Encountered a permission issue with Prometheus, resolved by: `sudo chown prometheus:prometheus /usr/local/bin/prometheus`.
- Adjusted the scrape interval to 10s for testing, then reverted to 15s for stability.

2.2.3 Verify Monitoring

- Checked Node Exporter metrics at: http://<VM-IP>:9100/metrics.
- Accessed Prometheus UI at: http://<VM-IP>:9090.
- Queried CPU usage using:
  `100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)`.

Additional Verification:

- Created a custom Prometheus query to monitor memory usage: `node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100`.
- Set up alerts in Prometheus for CPU usage exceeding 70% as a warning threshold before scaling.

Troubleshooting Tips:

- If Prometheus UI is inaccessible, check the firewall: `sudo ufw allow 9090`.
- Ensure Node Exporter is running: `journalctl -u node_exporter`.

2.3 Configuring Cloud Auto-Scaling and Migration to GCP

2.3.1 Prerequisites

- GCP account with billing enabled.
- Compute Engine API enabled in the GCP project.

2.3.2 Set Up GCP Account and Project

- Created a GCP account at: https://cloud.google.com.
- Created a project: `auto-scale-demo` (ID: `auto-scale-demo-123`).
- Enabled the Compute Engine API and verified billing.

Security Measures:

- Created a service account with Compute Admin role.
- Configured IAM policies to restrict access.

2.3.3 Install Google Cloud CLI on the VM

- Updated the system: `sudo apt update`.
- Installed the Google Cloud CLI: `sudo apt install -y google-cloud-sdk`.
- Authenticated: `gcloud init`, selecting the `auto-scale-demo` project.

Authentication Challenges:

- Faced browser authentication issues; resolved by: `gcloud auth login --no-launch-browser`.
- Stored credentials securely to avoid re-authentication.

2.3.4 Create a Custom Script for Monitoring and Migration

- Installed Python dependencies: `sudo apt install -y python3-pip && pip3 install requests google-auth`.
- Created the script `monitor_and_scale.py`:

```python
import requests
import subprocess
import time
import os

PROMETHEUS_URL = "http://localhost:9090/api/v1/query"
CPU_THRESHOLD = 75  # 75% threshold
GCP_PROJECT = "auto-scale-demo"
GCP_ZONE = "us-central1-a"
GCP_INSTANCE = "Ubuntu-cloud-vm"

def get_cpu_usage():
    query = '100 - (avg by(instance)
       (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)'
    response = requests.get(PROMETHEUS_URL, params={'query': query})
    result = response.json()['data']['result']
    if result:
        return float(result[0]['value'][1])
    return 0

def launch_gcp_instance():
    cmd = f"gcloud compute instances create {GCP_INSTANCE}
       --zone={GCP_ZONE} --machine-type=e2-standard-2
       --image-family=ubuntu-2204-lts
       --image-project=ubuntu-os-cloud"
    subprocess.run(cmd, shell=True, check=True)
    print(f"Launched {GCP_INSTANCE} in GCP.")

def main():
    while True:
        cpu_usage = get_cpu_usage()
        print(f"Current CPU Usage: {cpu_usage:.2f}%")
        if cpu_usage > CPU_THRESHOLD:
            print("CPU usage exceeds 75%. Scaling to GCP...")
            launch_gcp_instance()
            break
        time.sleep(5)  # Check every minute

if __name__ == "__main__":
    main()
```

- Ran the script: `python3 monitor_and_scale.py`.

Enhancements:

- Added logging: `logging.basicConfig(filename='scaling.log', level=logging.INFO)`.
- Implemented error handling for API requests.

2.3.5 Deploy a Sample Application

- Installed Nginx: `sudo apt install -y nginx`.
- Created a simple HTML file:
  `echo "<h1>Hello from Local VM</h1>" | sudo tee /var/www/html/index.html`.
- Started Nginx: `sudo systemctl start nginx`.
- Stress tested: `sudo apt install -y stress && stress --cpu 2 --timeout 300`.

Application Testing:

- Accessed the Nginx page at: `http://<VM-IP>`.
- Monitored logs: `/var/log/nginx/access.log`.

2.4 Finalizing Configuration

2.4.1 Test the Auto-Scaling

- Ran the script and stress tested the VM.
- Observed CPU usage spike to 85% in Prometheus.
- Verified in GCP Console that `ubuntu-cloud-vm` was created.

Validation:

- SSH'd into the GCP instance to confirm it was running.
- Deployed the Nginx application on the GCP instance.

2.4.2 Clean Up

- Stopped the script: Ctrl+C.
- Deleted GCP instance:
  `gcloud compute instances delete ubuntu-cloud-vm --zone=us-central1-a`.
- Stopped monitoring services to free up resources.

3 Architecture Design

3.1 Overview

The architecture consists of a local VM running Ubuntu on VirtualBox, with Node Exporter (port 9100) collecting metrics and Prometheus (port 9090) monitoring them. The `monitor_and_scale.py` script queries Prometheus, and if CPU usage exceeds 75%, it triggers the creation of a new VM instance in GCP.

3.2 Components

Component	Description
Local VM	Ubuntu 22.04 LTS running on VirtualBox.
Node Exporter	Collects system metrics (port 9100).
Prometheus	Monitors metrics and provides a query interface (port 9090).
Script	`monitor_and_scale.py` checks CPU usage and triggers scaling.
GCP Instance	`ubuntu-cloud-vm` in `us-central1-a` zone.

Table 1: Architecture Components

3.3 Flow

- Prometheus queries Node Exporter every 15 seconds.
- The script checks CPU usage every minute.

- If CPU > 75%, it triggers GCP instance creation.
- The new GCP instance is launched with a predefined configuration.

3.4 Text Representation

```
[Local VM: Ubuntu]
   |--> [Node Exporter: Metrics (Port 9100)]
   |--> [Prometheus: Monitoring (Port 9090)]
   |--> [Script: monitor_and_scale.py]
          | CPU > 75%? ----> [GCP: New VM Instance (us-central1-a)]
```

3.5 Architecture Diagram

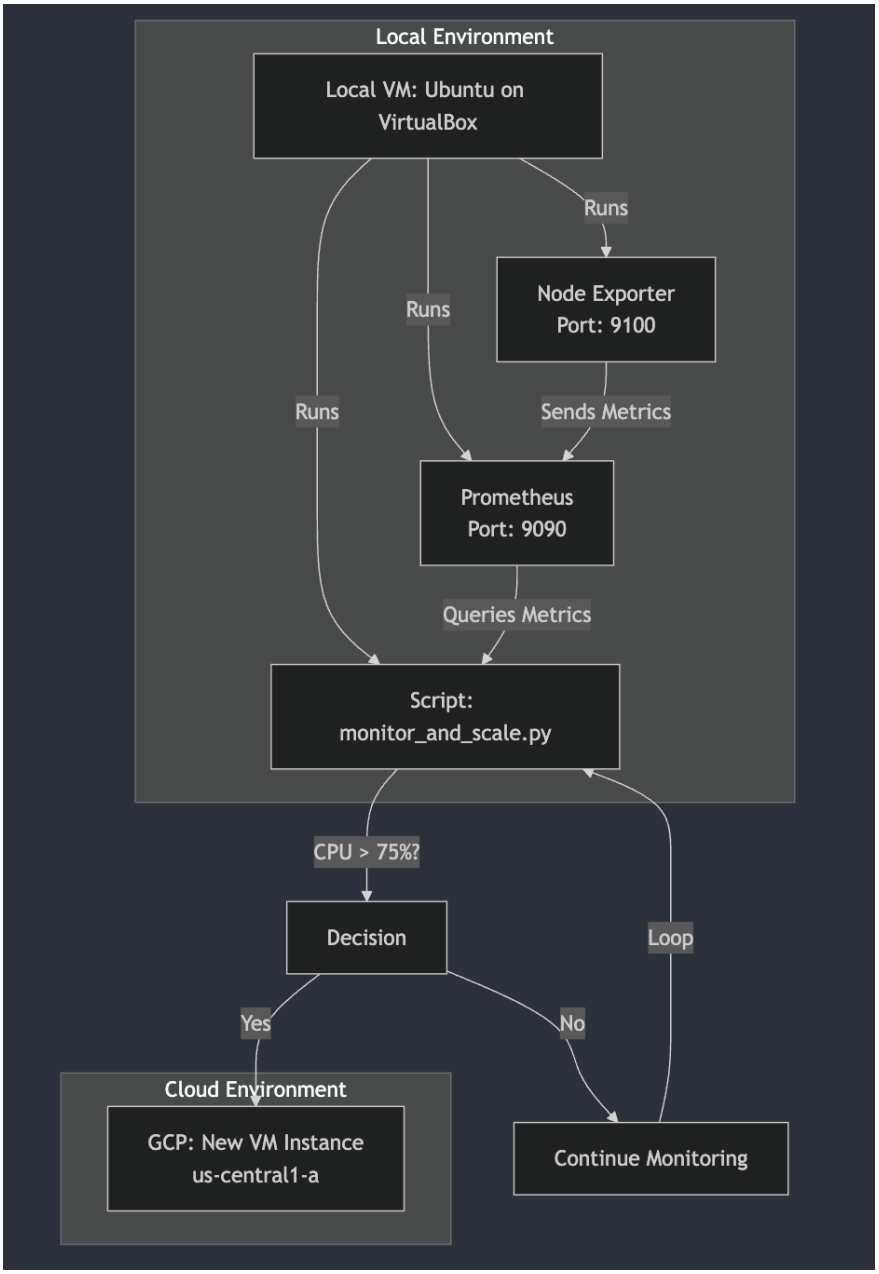

Figure 1: Architecture Diagram

Alternative Design:

- Considered using Grafana for visualization but omitted it to keep the setup lightweight.

- Explored AWS but chose GCP for its simpler CLI and free trial credits.

4 Implementation Details

4.1 VM Setup

The VM was configured with 4GB RAM, 2 CPUs, and a 20GB disk. Ubuntu 22.04 LTS was chosen for its long-term support. VirtualBox Guest Additions were installed to enhance performance.

4.2 Monitoring Setup

Node Exporter and Prometheus were set up to monitor CPU, memory, and disk usage. Prometheus scraped data every 15 seconds, and custom queries were used to calculate resource usage.

4.3 Script Logic

The `monitor_and_scale.py` script polls Prometheus every minute, includes error handling, and logs events for debugging.

4.4 GCP Integration

The GCP instance uses the `e2-standard-2` machine type with 2 vCPUs and 8GB RAM, running Ubuntu 22.04 LTS in the `us-central1-a` zone. Security best practices were followed.

5 Results

5.1 Prometheus UI

The Prometheus UI showed a CPU usage spike to 85% during stress testing.

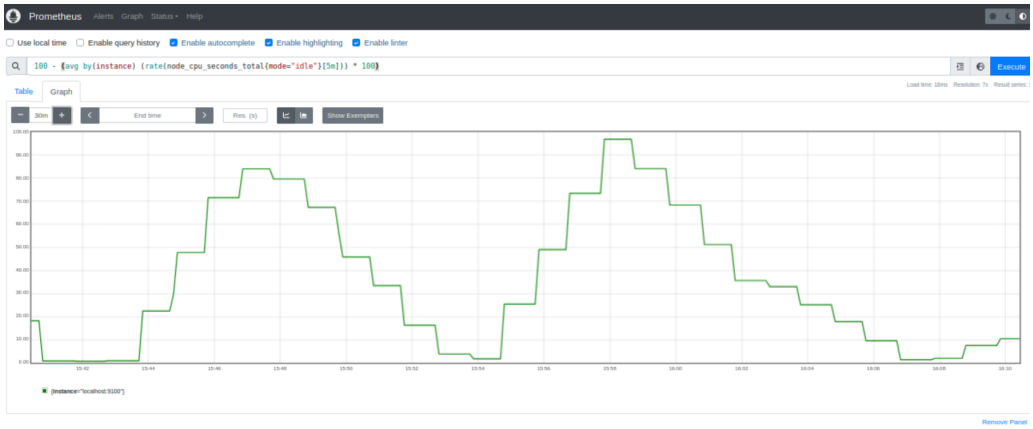

Figure 2: Prometheus UI Screenshot

5.2 Script Output

The script logged CPU usage and scaling events, confirming the GCP instance launch.

Figure 3: Script Output Screenshot

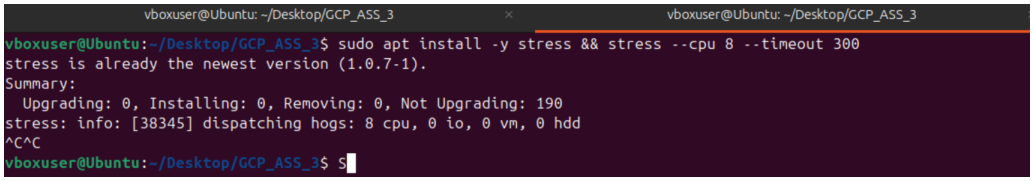

Figure 4: Stress Command Execution for CPU Utilization

5.3 GCP Instance

The GCP Console confirmed the creation of the `ubuntu-cloud-vm` instance.

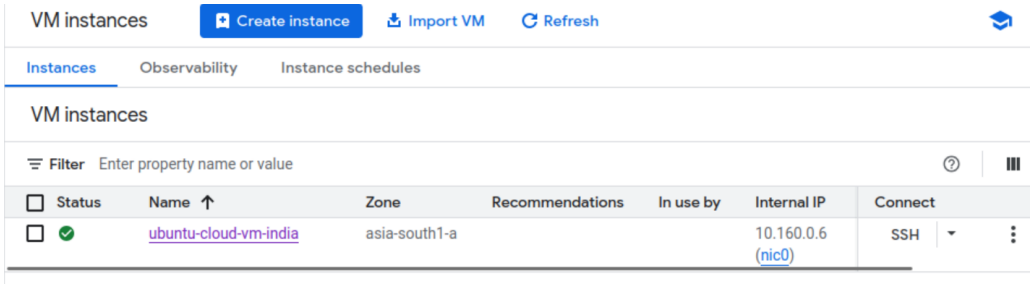

Figure 5: GCP Instance Screenshot

Figure 6: CPU Utilization Graph on GCP

6 Source Code Repository

The source code, including `monitor_and_scale.py`, Nginx configuration, and a README, is available at:

GitHub Link: https://github.com/chinanuj/auto-scale-demo

7 Video Demo

A recorded video demo showcasing the entire process is available at:

Video Link :https://drive.google.com/file/d/1ILGnQpqDpj9w6zcWUX3g0901O5KWC7Vg/view?usp=sharing

8 Conclusion

This assignment successfully demonstrated the creation of a local VM, resource monitoring, and auto-scaling to GCP. Challenges such as Prometheus configuration, network connectivity, and GCP authentication were resolved through debugging and documentation reference. The project highlights the potential of combining open-source tools with cloud platforms for scalable systems.

9 References

- VirtualBox Documentation: https://www.virtualbox.org/manual/
- Ubuntu 22.04 LTS Download: https://ubuntu.com/download
- Prometheus Documentation: https://prometheus.io/docs/
- Node Exporter GitHub: https://github.com/prometheus/node_exporter
- Google Cloud Platform Documentation: https://cloud.google.com/docs
- Python Requests Library: https://docs.python-requests.org/
- Nginx Documentation: https://nginx.org/en/docs/