Auto-Scaling Compute Resources:

A Real-World Demonstration Using Local VM and Google Cloud

# Introduction

In many real-world systems, workloads begin on small, local machines — virtual environments, low-power servers, or edge devices. These systems are cost-effective for light processing but cannot sustain high computing demands indefinitely. We need a scalable, on-demand solution that avoids permanent cloud infrastructure costs as resource usage spikes.

This project was designed to solve that challenge. We created a working system that:

* Starts computation on a local Alpine Linux virtual machine.
* Monitors CPU and RAM usage.
* Triggers GCP VM auto-scale only when required.
* Automatically stops local load post-scale.

# Objective

* Build a lightweight, local VM setup for computation.
* Monitor real-time CPU and RAM usage from the host.
* Trigger GCP auto-scale when usage exceeds 75%.
* Automatically stop local load process after scale-up.
* Keep the setup fully scriptable, cost-friendly.

# Tools & Technologies

* VirtualBox: Local VM creation
* Alpine Linux: Lightweight, fast-booting OS
* Python 3: Monitoring and automation scripts
* Paramiko: SSH automation from host to VM
* NumPy: For matrix-based CPU load simulation
* Google Cloud Platform (GCP): Cloud compute resource
* gcloud CLI: Command-line access to GCP services
* Windows Host: Control layer for monitoring and scaling

# Architecture Overview

The system follows a lightweight, modular architecture suitable for real-time compute offloading from a local VM to Google Cloud Platform (GCP). Here's how the architecture works:

* + The local computation starts in an Alpine Linux VM running on VirtualBox. This VM executes matrix computations via a Python script (`matrix\_cpu\_load.py`) to simulate CPU load.
  + A monitoring script (`monitor.py`) checks the CPU and RAM usage from inside the Alpine VM. This script is triggered remotely from the host system using SSH.
  + The host system (e.g., a Windows PC) runs a control script (`trigger\_gcp\_vm.py`). This script uses Paramiko to SSH into the VM, runs `monitor.py`, and retrieves CPU and RAM usage.
  + If the CPU or RAM usage exceeds 75%, the control script uses the Google Cloud CLI (`gcloud`) to start a pre-configured GCP VM instance. This mimics horizontal scaling.
  + Once the GCP instance is started, the control script also remotely stops the CPU load running on the local VM to ensure cost-efficiency and prevent duplicate processing.

A diagram of a computer process

AI-generated content may be incorrect.

Figure: Auto-Scaling Architecture from Local VM to GCP

# Project Setup and Flow

## Local Virtual Machine (Alpine)

* + Installed via ISO in VirtualBox
  + Basic packages and Python3 installed
  + SSH enabled and accessible from host
  + Files transferred using scp

## Resource Monitoring (monitor.py)

* + Resides in Alpine VM
  + Reads CPU stats from /proc/stat (idle time difference)
  + Reads RAM stats from /proc/meminfo (MemTotal vs. MemAvailable)
  + Reports as:

CPU Usage: 92.3%

RAM Usage: 43.7%

## CPU Load Simulation (matrix\_cpu\_load.py)

* + Runs an infinite loop of matrix multiplications using NumPy
  + Keeps CPU usage consistently high
  + Triggered remotely using:

ssh root@<VM-IP> "nohup python3 ~/matrix\_cpu\_load.py > /dev/null 2>&1 &"

## GCP Setup

* + Pre-created VM instance (my-vm-instance) in us-central1-c
  + GCP project and billing already configured
  + gcloud SDK installed and configured on host
  + GCP VM is kept in a stopped state, only triggered when needed

# Core Controller Script (trigger\_gcp\_vm.py)

* + SSH into Alpine VM using Paramiko
  + Executes monitor.py and captures output
  + Parses CPU and RAM values
  + If usage exceeds 75%, executes:

gcloud compute instances start my-vm-instance --zone=us-central1-c

* + Also runs a cleanup command:

pkill -f matrix\_cpu\_load.py

to stop the local CPU load simulation once scale-up is triggered

# Demo Walkthrough

1. Start Alpine VM in VirtualBox

* + Ensure networking is functional
  + SSH into Alpine VM from host to confirm access

2. Start matrix load

* + From Windows CMD:

ssh root@192.168.56.110 "nohup python3 ~/matrix\_cpu\_load.py > /dev/null 2>&1 &"

3. Run trigger script

* + From host system:

python trigger\_gcp\_vm.py

* + Output:

CPU Usage: 93.2%

RAM Usage: 45.0%

* Triggering GCP scale-up...
* GCP VM started successfully.
* CPU load stopped.

4. Verify on GCP Console

* + Navigate to Compute Engine > VM Instances
  + Observe status change: TERMINATED → RUNNING

5. Post-demo cleanup

* + From GCP Console or CLI:

gcloud compute instances stop my-vm-instance --zone=us-central1-c

# Real-World Relevance

This setup closely mirrors production edge-device offloading. Lightweight nodes process until overwhelmed. Once thresholds are hit, workloads scale to the cloud. Our approach adds:

* + SSH-based control without agents
  + Cloud-triggering via thresholds
  + Post-scale cleanup to manage costs

# File Summary

|  |  |
| --- | --- |
| **File Name** | **Description** |
| monitor.py | Runs in Alpine, returns CPU and RAM usage |
| matrix\_cpu\_load.py | Infinite matrix multiplication loop |
| trigger\_gcp\_vm.py | Runs on host, monitors VM and scales GCP |