Assignment 2: Use GCP Cloud to Create a VM to Leverage Auto Scaling and Security

Shivanshu Verma
B22ES010
Course: Virtualization and Cloud Computing
Indian Institute of Technology, Jodhpur

02-03-2023

Abstract

This report documents the process of setting up a virtual machine (VM) in Google Cloud Platform (GCP) that leverages auto-scaling policies and enhanced security measures. The implemented solution includes the creation of a backend application designed to generate CPU load, containerizing the application using Docker, and deploying it via an instance template and managed instance group. A load balancer named `autoscale` is configured to distribute traffic. Additionally, firewall rules and IAM roles are established to ensure secure and scalable operations.

Contents

Repository and Video Demonstration Links

- **Source Code Repository:** `https://github.com/Shivanshu-Verma/VCC-A2`
- **Video Demonstration:** `View Demonstration`

1 Introduction

The primary objective of this assignment is to create a scalable and secure infrastructure on GCP. The system utilizes an auto-scaling policy that monitors CPU usage and dynamically adjusts the number of VM instances. A backend server application, which calculates the Fibonacci sequence recursively for large inputs, is used to simulate a high CPU load. This leads to the triggering of the auto-scaling mechanism. The solution also involves configuring firewall rules to allow HTTP traffic (port 80) and applying appropriate IAM roles for access control.

2 Methodology

The overall approach involves the following steps:

1. **Backend Application Development:** A server application is developed that calculates Fibonacci numbers recursively. When a POST request is sent with parameter `n`, the application performs a heavy recursive computation, thereby creating CPU load.
2. **Containerization:** The application is dockerized and the image is made publicly available on Docker Hub.
3. **Automation Script:** A startup script is created for each VM instance to install Docker, pull the Docker image, and run the container.
4. **Instance Template Creation:** An instance template is configured (using an `e2-micro` instance) that incorporates the startup script.
5. **Managed Instance Group:** An autoscaling instance group is created based on the instance template.
6. **Load Balancer Configuration:** A regional, external load balancer named `autoscale` is set up to route traffic to the instance group.
7. **Security Measures:** Firewall rules are applied to allow HTTP traffic on port 80, and IAM roles are set up to restrict access.

3 Implementation Details

3.1 Backend Application and Containerization

A backend server application was developed to compute the Fibonacci sequence recursively. The application is designed to accept a POST request with a parameter `n`, where a large value of `n` results in high CPU usage.

The application was then dockerized. The Docker image is available publicly on Docker Hub at `https://hub.docker.com/r/shivanshu010/fibonacci/tags`.

3.2 Startup Automation Script

The following bash script is used as the startup script for each VM instance. It installs Docker, starts the Docker service, and pulls and runs the Docker image:

```bash
#!/bin/bash
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker

# Pull and run the Docker image from Docker Hub
sudo docker run -d -p 80:5000 shivanshu010/fibonacci:latest
```

Listing 1: Startup Script for VM Instances

3.3 Instance Template Setup

An instance template named `autoscale-template` was created using an `e2-micro` machine type. Key configuration parameters include:

- **Operating System and Disk:** The template uses a Debian 12 (Bookworm) image (`projects/debian-` `/global/images/debian-12-bookworm-v20250212`) with a 10 GB persistent balanced disk (`pd-balanced`). The disk is set to auto-delete upon instance deletion.

- **Network and Tags:** The instance is attached to the default network with a single network interface that uses an external NAT configuration. It is tagged with `http-server`, `https-server`, and `lb-health-check` to facilitate firewall and load balancing rules.

- **Scheduling and Maintenance:** The scheduling configuration ensures that instances will `MIGRATE` during host maintenance, have automatic restarts enabled, and are not preemptible. The provisioning model is set to `STANDARD`.

- **Startup Script:** A metadata item contains the startup script that updates packages, installs Docker, starts the Docker service, and then pulls and runs the Docker image `shivanshu010/fibonacci:latest` on port 80.

- **Security and Service Accounts:** Shielded VM configurations are enabled for integrity monitoring and virtual TPM support, while secure boot is disabled. The instance uses a service account with scopes for read-only storage, logging, monitoring, and service control.

3.4 Managed Instance Group Configuration

Using the `autoscale-template` instance template, a managed instance group named `autoscale-group` was created in zone `asia-south2-a`. The configuration details include:

- **Instance Group Properties:** The instance group initially has one instance, with a named port `api` mapped to port 80. The group is attached to the default network and subnet in the `asia-south2` region.
- **Autoscaling Policy:**
  - **CPU Utilization Target:** 60% (0.6 target utilization).
  - **Scaling Limits:** Minimum number of replicas is set to 1 and maximum to 3.
  - **Autoscaler Mode:** The autoscaler is active and configured with a cooldown period of 30 seconds between scaling actions.
- **Instance Group Manager:** The group is managed by an instance group manager which references the `autoscale-template`. It uses an update policy with minimal actions set to `REPLACE`, and both `maxSurge` and `maxUnavailable` are fixed to 1 to maintain service stability during updates.

3.5 Load Balancer Setup

A regional, external load balancer named `autoscale` was configured with the following details:

- **Frontend Configuration:** The load balancer listens for HTTP traffic on port 80 with no session affinity. The connection draining timeout is set to 300 seconds.
- **Backend Service:**
  - The load balancer uses the `autoscale-group` as its backend, with the named port `api` (port 80) designated for traffic.
  - Balancing mode is set to `UTILIZATION` with a maximum utilization threshold of 80% and a capacity scaler of 1.
- **Timeout and Health Checks:** The load balancer has a timeout setting of 600 seconds and utilizes a health check (referenced as `autoscale`) to ensure backend instances are healthy.
- **Additional Properties:** The load balancing scheme is set to `EXTERNAL_MANAGED` with a locality policy of `ROUND_ROBIN`. A default security policy is also applied to safeguard backend service communications.

3.6 Firewall Rules and IAM Configuration

Firewall Rule: A firewall rule was created to allow incoming HTTP traffic (port 80) from all sources to all VM instances.
IAM Roles: Appropriate IAM roles were set up to ensure restricted access to the instances, thereby enforcing a secure operational environment.

4 Testing and Results

To validate the auto-scaling functionality, the following steps were performed:

1. The load balancer's public IP **34.131.12.116** was obtained.
2. A frontend application sent a POST request to the endpoint:

   `http://34.131.12.116:80/fibonacci`

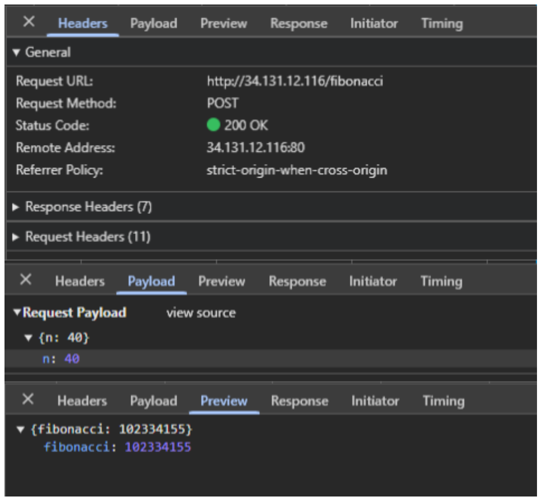
   with `n = 40` as the parameter.
3. The backend application computed the Fibonacci sequence recursively, causing the CPU usage on the initial VM instance to spike.
4. The configured auto-scaling policy detected the CPU utilization exceeding the 60% threshold and automatically spun up two additional VM instances (for a total of three).

The system behaved as expected:

- Traffic was balanced across all VM instances via the load balancer.
- Instances were automatically added or removed based on the CPU load in real-time.
- Security rules (firewall and IAM) ensured that only authorized traffic and access were permitted.

4.1 Load Balancer Request and Response

Figure 1 provides an example of the request and response when sending `n=40` to the load balancer endpoint. This demonstrates how the backend application is triggered to perform the Fibonacci computation.



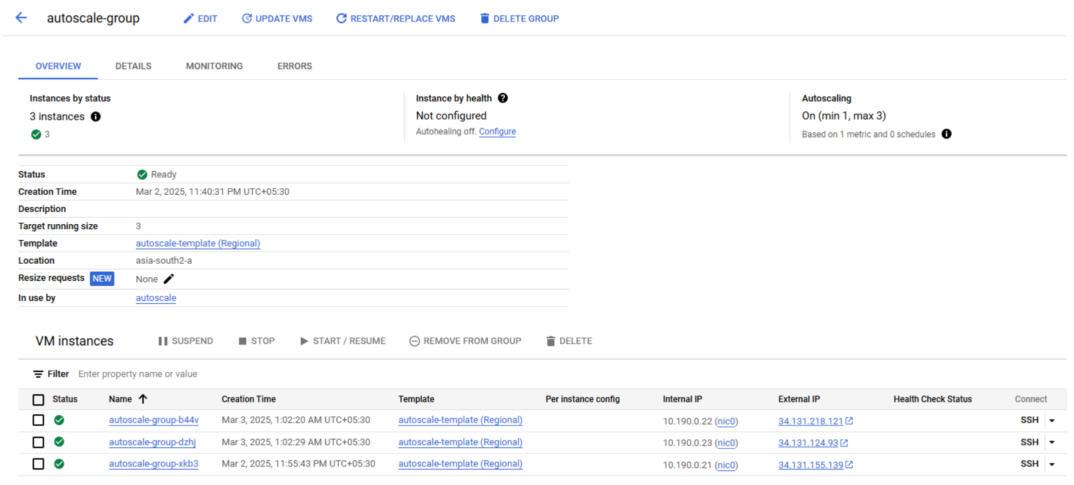
```
Request URL:      http://34.131.12.116/fibonacci
Request Method:   POST
Status Code:      ● 200 OK
Remote Address:   34.131.12.116:80
Referrer Policy:  strict-origin-when-cross-origin
```

```
▾Request Payload
 ▾ {n: 40}
    n: 40
```

```
▾ {fibonacci: 102334155}
   fibonacci: 102334155
```

Figure 1: Request to `http://34.131.12.116:80/fibonacci` and corresponding response

4.2 Google Cloud Console Screenshots



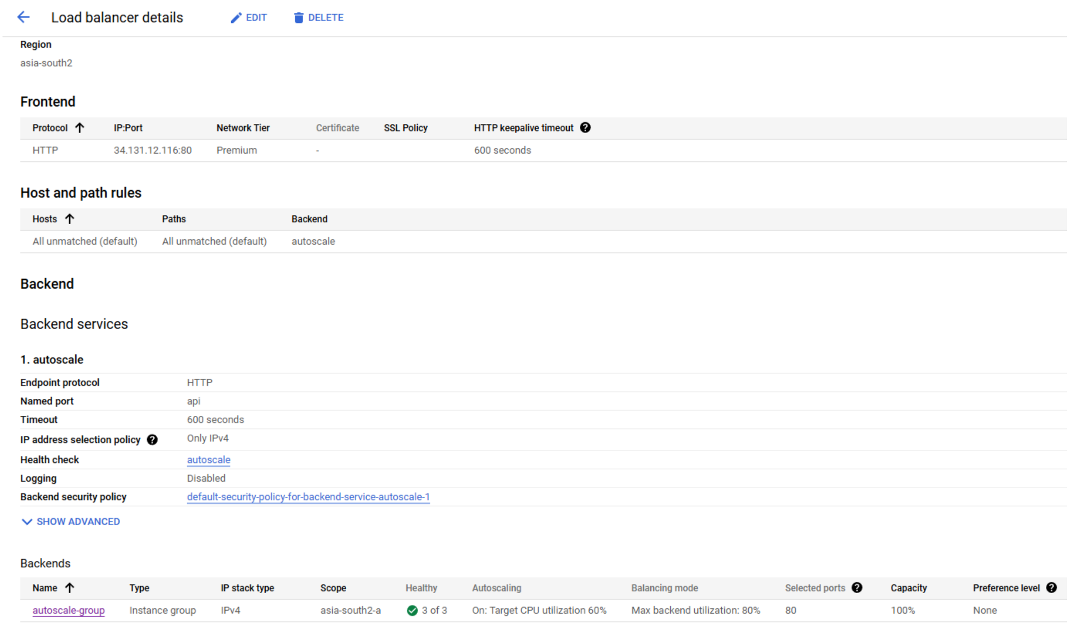
Instances by status	Instance by health	Autoscaling
3 instances	Not configured	On (min 1, max 3)
✓ 3	Autohealing off. Configure	Based on 1 metric and 0 schedules

Status	✓ Ready
Creation Time	Mar 2, 2025, 11:40:31 PM UTC+05:30
Description	
Target running size	3
Template	autoscale-template (Regional)
Location	asia-south2-a
Resize requests NEW	None
In use by	autoscale

VM instances

Status	Name	Creation Time	Template	Per instance config	Internal IP	External IP	Health Check Status	Connect
✓	autoscale-group-b44v	Mar 3, 2025, 1:02:20 AM UTC+05:30	autoscale-template (Regional)		10.190.0.22 (nic0)	34.131.218.121		SSH
✓	autoscale-group-dzhj	Mar 3, 2025, 1:02:29 AM UTC+05:30	autoscale-template (Regional)		10.190.0.23 (nic0)	34.131.124.93		SSH
✓	autoscale-group-xkb3	Mar 2, 2025, 11:55:43 PM UTC+05:30	autoscale-template (Regional)		10.190.0.21 (nic0)	34.131.155.139		SSH

Figure 2: Instance Group in Google Cloud Console showing the autoscaling VMs



Region
asia-south2

Frontend

Protocol	IP:Port	Network Tier	Certificate	SSL Policy	HTTP keepalive timeout
HTTP	34.131.12.116:80	Premium	-		600 seconds

Host and path rules

Hosts	Paths	Backend
All unmatched (default)	All unmatched (default)	autoscale

Backend

Backend services

1. autoscale

Endpoint protocol	HTTP
Named port	api
Timeout	600 seconds
IP address selection policy	Only IPv4
Health check	autoscale
Logging	Disabled
Backend security policy	default-security-policy-for-backend-service-autoscale-1

Backends

Name	Type	IP stack type	Scope	Healthy	Autoscaling	Balancing mode	Selected ports	Capacity	Preference level
autoscale-group	Instance group	IPv4	asia-south2-a	✓ 3 of 3	On: Target CPU utilization 60%	Max backend utilization: 80%	80	100%	None

Figure 3: Load Balancer Configuration in Google Cloud Console

These console views (Figures 2 and 3) confirm that the managed instance group is successfully scaling the number of instances according to the defined policy, and that the load balancer is properly distributing traffic across these instances.

5 Architecture Diagram

Figure 4 illustrates the high-level design of the load-balanced, managed, and autoscaling infrastructure on Google Cloud Platform (GCP). The complete diagram can be viewed using the provided draw.io link. The diagram highlights the following key components and interactions:

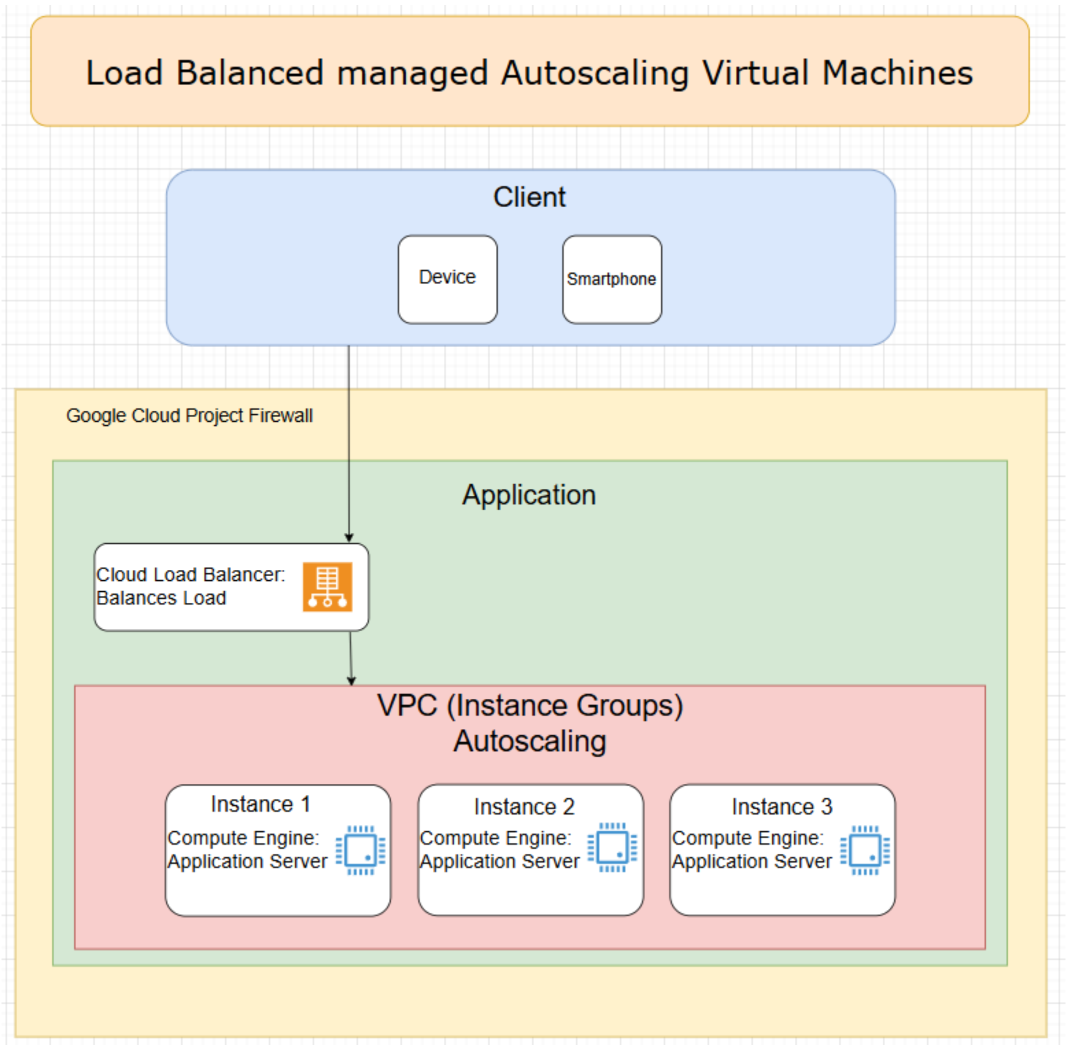

Load Balanced managed Autoscaling Virtual Machines
Client
Device
Smartphone
Google Cloud Project Firewall
Application
Cloud Load Balancer: Balances Load
VPC (Instance Groups) Autoscaling
Instance 1
Compute Engine: Application Server
Instance 2
Compute Engine: Application Server
Instance 3
Compute Engine: Application Server

Figure 4: High-Level Architecture of Load-Balanced, Autoscaling VMs on GCP

1. **Client Layer:** End-users (on devices such as laptops, smartphones, etc.) send requests over the internet to the application.

2. **Google Cloud Project Firewall:** The GCP firewall enforces security rules. In this setup, it allows incoming traffic on port 80 (HTTP) to reach the load balancer and subsequently the instances.

3. **Cloud Load Balancer:** Acts as the single entry point for all external traffic. It receives client requests and distributes them across the backend instances based on load and health checks.

4. **VPC (Instance Groups) Autoscaling:**

    - **Managed Instance Group (MIG):** A group of virtual machine instances created from a common instance template. The MIG automatically scales the number of instances up or down based on CPU utilization.
    - **Compute Engine Instances:** Each instance runs the containerized application (the Fibonacci service). As the load increases and CPU usage crosses the configured threshold (60%), additional instances are spawned (up to a maximum of 3). When CPU usage remains below the threshold for a defined cooldown period, the group scales back down.

5. **Application Layer:** Each Compute Engine instance hosts the backend service (Fibonacci calculation), which is exposed via HTTP on port 80. Docker is used to run the application container.

This design ensures that traffic is evenly distributed, while the environment can dynamically handle varying loads by automatically adjusting the number of active VM instances. The firewall configuration, combined with the load balancer's health checks and autoscaling policies, provides both security and scalability for the application.

6 Conclusion

In this assignment, a scalable and secure cloud infrastructure was successfully implemented on GCP. The solution demonstrated:

- The ability to handle increased workloads via auto-scaling based on CPU utilization.
- Secure operations through firewall configurations and IAM role assignments.
- Seamless integration of containerized applications using Docker.

The final system not only meets the requirements set forth in the assignment but also provides a robust framework that can be extended for further enhancements.