Applied Research

Selenium in Cloud Environments on Google Cloud Platform (GCP)

Verma Mohit

Matriculation ID: 11027900

SRH Hochschule Heidelberg

Supervisor:

Prof. Paul Tanzer

Table of Contents

1. Introduction

Automated testing is a vital component of modern software development, enabling continuous integration and rapid feedback. Web application testing frequently utilizes Selenium, an open-source test automation framework. This research examines the deployment of Selenium Grid on Google Cloud Platform (GCP) using Virtual Machines (VMs) alongside Jenkins for Continuous Integration and Continuous Deployment (CI/CD). The objective is to optimize security, cost efficiency, and performance while ensuring scalable and reliable test execution.

2. Research Objectives & Questions

2.1 Deployment & Configuration

- How can Selenium Grid be effectively deployed and configured on GCP (e.g., Compute)
  - Selenium Grid can be deployed on GCP using Compute Engine (VMs) for a straightforward setup. The preferred method depends on test volume, required parallelism, and ease of management.
- How can we monitor, measure, and optimize Selenium test execution performance in cloud environments?
  - GCP's Cloud Monitoring and Logging services, combined with Prometheus and Grafana, provide insights into test execution performance. Key optimizations include tuning request/response times, reducing unnecessary browser instantiations, and optimizing test scripts.

- **How can we monitor, measure, and optimize Selenium test execution performance in cloud environments?**
  - GCP's Cloud Monitoring and Logging services, combined with Prometheus and Grafana, provide insights into test execution performance. Key optimizations include tuning request/response times, reducing unnecessary browser instantiations, and optimizing test scripts.

3. Technologies & Tools

3.1 Selenium

- **Selenium WebDriver** – Core library for automating web browsers.
- **Selenium Grid** – Enables distributed, parallel test execution.

3.2 Google Cloud Platform (GCP)

- **Compute Engine (VMs)** – Traditional virtual machines for Selenium deployments.
- **Cloud Monitoring & Logging** – Ensures observability and diagnostics.

3.3 Containerization & CI/CD

- **Docker** – Packages Selenium and test environments for consistent deployment.
- **Jenkins / GitLab CI / GitHub Actions** – Integrates automated tests into CI/CD pipelines.

3.4 Programming Languages

- **Python** – Commonly used for Selenium-based test automation.

4. Expected Challenges & Mitigation Strategies

4.1 Network Latency & Performance Bottlenecks

Issue: High latency can impact test execution speeds.
Solution: Use regional GCP instances close to application servers and optimize network configurations.

4.2 Resource Scaling & Management

Issue: Managing test execution for high concurrency and parallelism.
Solution: Implement Kubernetes autoscaling, manage worker nodes efficiently, and use spot instances for cost reduction.

4.3 Security & Compliance

Issue: Data security risks when running tests in the cloud.
Solution: Use secure firewall settings, encryption, and IAM roles for access control.

5. Understanding Selenium Grid

5.1 What is Selenium Grid?

Selenium Grid is a component of the Selenium testing framework that enables test execution across multiple machines, browsers, and environments. It supports parallel execution, reducing test execution time.

- **Hub:** The central server that distributes test execution requests to registered nodes.
- **Nodes:** Machines that execute test cases on different browsers and operating systems.

5.2 Benefits of Selenium Grid

- **Parallel Execution:** Run multiple tests across different browsers, versions, and OS simultaneously.
- **Reduced Execution Time:** Distributes test cases across multiple nodes to speed up execution.
- **Cross-Browser Testing:** Ensures compatibility testing across different browser versions.
- **Remote Execution:** Optimizes resource utilization by running tests on remote machines.
- **Scalability:** Easily adds new nodes dynamically to the test infrastructure.

6. Overview of Google Cloud Platform (GCP) for Testing

6.1 What is Compute Engine?

Google Compute Engine (GCE) is an Infrastructure-as-a-Service (IaaS) offering that provides virtual machines (VMs) on demand. It supports various machine types, storage options, and networking configurations.

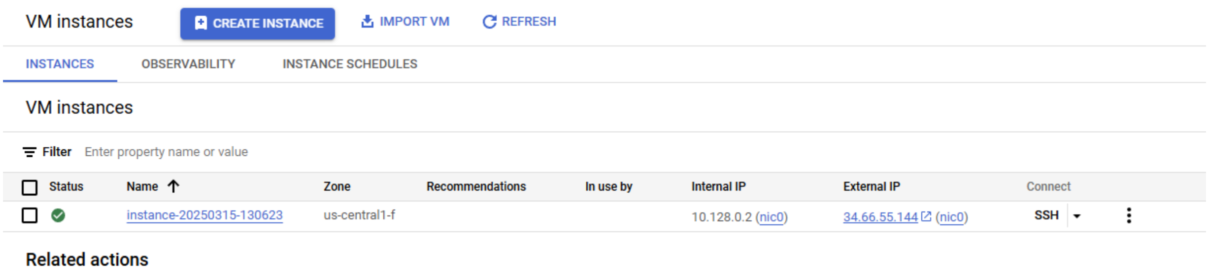
6.2 Use Cases for Selenium Testing

- Running headless browsers for UI testing.
- Deploying Selenium Grid for distributed test execution.
- Integrating with CI/CD pipelines for automated testing.

7. Setting Up Selenium Grid with Docker on Google Cloud Virtual Machines

7.1 Creating a Virtual Machine Using GCP Console

1. Navigate to **Google Cloud Console**.
2. Select **Compute Engine > VM Instances**.
3. Click **Create Instance**.
4. Configure instance details:
   - **Name:** instance-20250315-130623
   - **Region:** Choose the nearest region
   - **Machine type:** n1-standard-2
   - **Boot disk:** Ubuntu 20.04 LTS
5. Click **Create** to launch the instance.



□	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
□	✓	instance-20250315-130623	us-central1-f			10.128.0.2 (nic0)	34.66.55.144 (nic0)	SSH

Fig. VM Instance Dashboard

7.2 Creating a Docker Compose File

Create a `docker-compose.yaml` file to define Selenium Hub and Nodes:

```
services:
  chrome:
    image: selenium/node-chrome
    depends_on:
      - hub
  firefox:
    image: selenium/node-firefox
    depends_on:
      - hub
  hub:
    image: selenium/hub
    ports:
      - 4444:4444
```

8. Deploying Selenium Grid with Docker Compose

8.1 Install Docker and Docker Compose:

```
sudo apt-get update

sudo apt install docker.io
```

8.2 Start and Enable Docker

```
sudo systemctl start docker

sudo systemctl enable docker
```

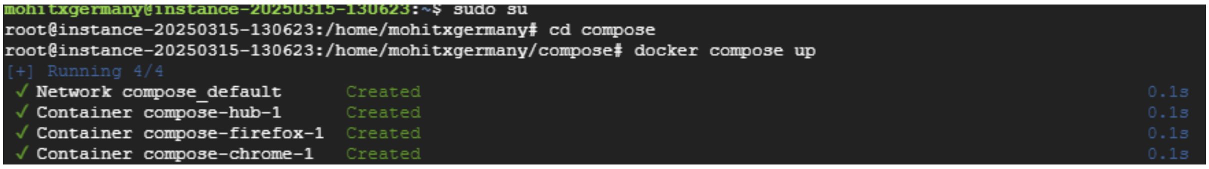
8.3 Install Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

8.4 Navigate to the directory containing docker-compose.yml and run:

```
docker-compose up
```

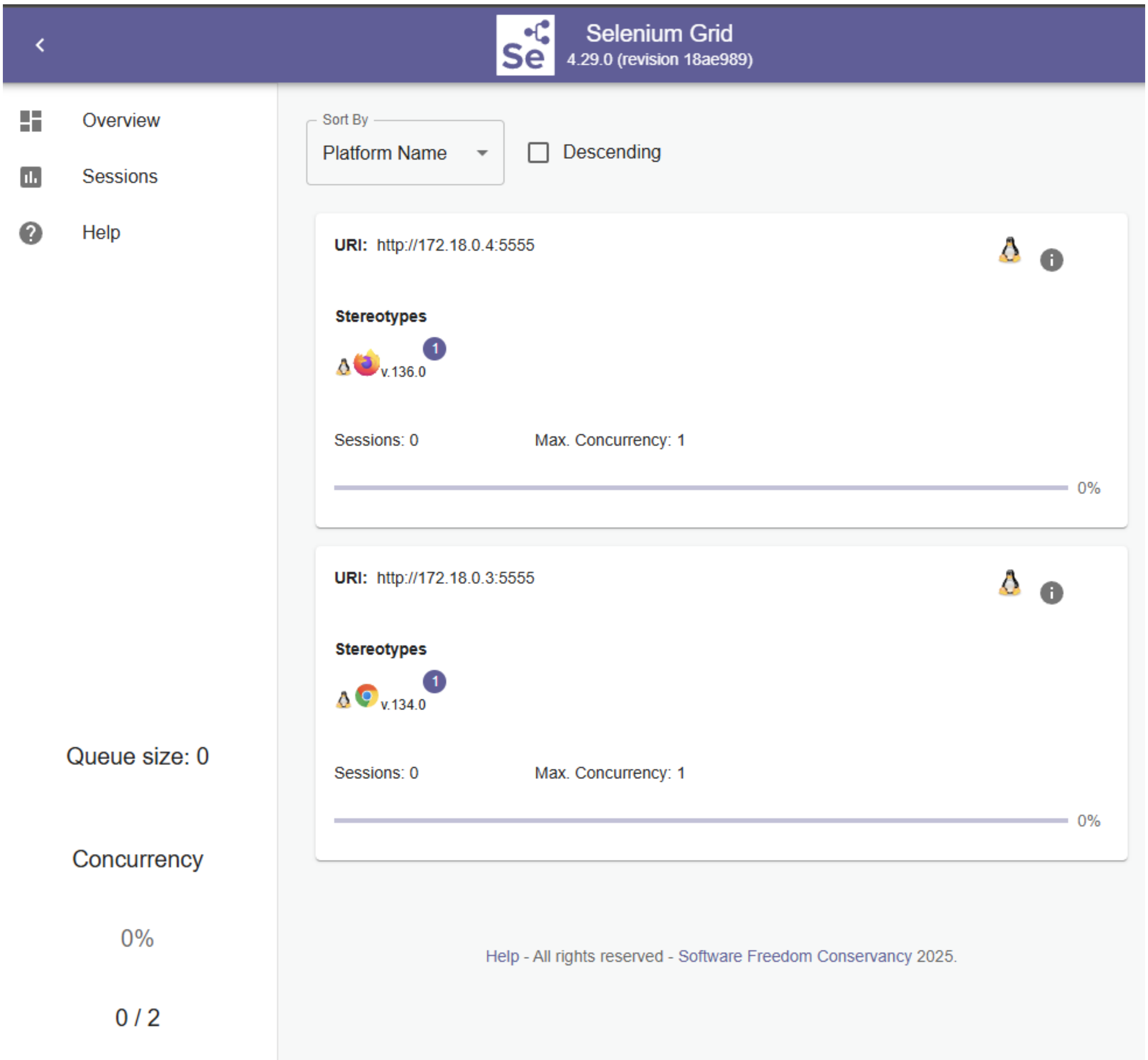


```
mohitxgermany@instance-20250315-130623:~$ sudo su
root@instance-20250315-130623:/home/mohitxgermany# cd compose
root@instance-20250315-130623:/home/mohitxgermany/compose# docker compose up
[+] Running 4/4
 ✓ Network compose_default       Created                                                                0.1s
 ✓ Container compose-hub-1       Created                                                                0.1s
 ✓ Container compose-firefox-1   Created                                                                0.1s
 ✓ Container compose-chrome-1    Created                                                                0.1s
```

Fig. After Composing Hub and Node Containers

8.5 Verify Selenium Grid Setup

Access the Selenium Grid Console at http://<VM_IP>:4444/grid



Selenium Grid
4.29.0 (revision 18ae989)

Sort By: Platform Name
□ Descending

URI: http://172.18.0.4:5555
Stereotypes
v.136.0 (1)
Sessions: 0   Max. Concurrency: 1
0%

URI: http://172.18.0.3:5555
Stereotypes
v.134.0 (1)
Sessions: 0   Max. Concurrency: 1
0%

Queue size: 0

Concurrency

0%

0 / 2

Help - All rights reserved - Software Freedom Conservancy 2025.

Fig. Selenium Hub

9. Sample Selenium Test Script (Python)

Save the following script as **selenium_grid_test.py** on your local machine:

```
from selenium import webdriver
import time
options = webdriver.ChromeOptions()
options.add_argument("--headless")

driver = webdriver.Remote(
    command_executor="http://35.202.64.238:4444/wd/hub",
    options=options
)


driver.get("https://www.google.com")
print("Page Title:", driver.title)

driver.quit()
time.sleep(5)
```

10. Setting Up Jenkins for Selenium Testing

10.1 Install Jenkins on Your Local Machine

For **Windows**:

1. Download and install Jenkins from Jenkins official website.
2. Start Jenkins and access it at: http://localhost:8080/.

10.2 Start Jenkins

```
sudo systemctl start jenkins

sudo systemctl enable jenkins
```

Access Jenkins via:

```
http://localhost:8080/
```

Retrieve the **admin password**:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

11. Configure Jenkins to Run a Local Selenium Project

11.1 Install Required Plugins

- **Pipeline Plugin**
- **Git Plugin** (even though the project is local)

1. Navigate to **Manage Jenkins > Plugins > Available Plugins**.
2. Install the above plugins and restart Jenkins.

12. Create a Jenkins Job for Your Local Project

12.1 Create a Freestyle Job

1. **Go to Jenkins Dashboard > New Item > Freestyle Project**.
2. **Enter a Name** (“`selenium_grid_on_gcp`").
3. **In "Build Triggers"**
4. **In "Build Environment"**, check **"Use a custom workspace"** and enter the path to your local Selenium project:

```
    C:\Users\ROG\OneDrive\Documents\GitHub\SeleniumProject
(Windows)
```

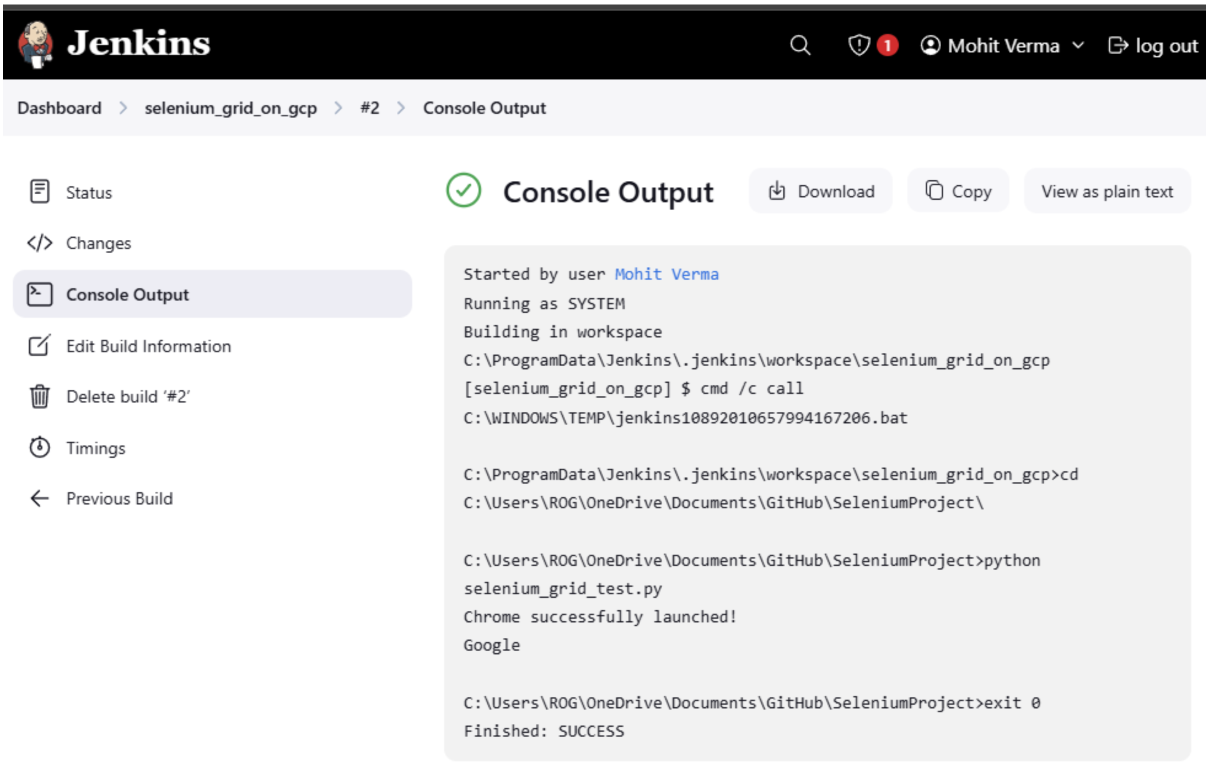
12.2 Add Build Commands

Under **"Build Steps" > "Execute Shell" (Linux/Mac)** or **"Execute Windows Batch Command" (Windows)**:

For **Windows** (Batch command):

```
cd C:\Users\ROG\OneDrive\Documents\GitHub\SeleniumProject\
python selenium_grid_test.py
```

5. Click **Save & Build Now**.
6. Click on the latest Build and click on the console output:



```
Started by user Mohit Verma
Running as SYSTEM
Building in workspace
C:\ProgramData\Jenkins\.jenkins\workspace\selenium_grid_on_gcp
[selenium_grid_on_gcp] $ cmd /c call
C:\WINDOWS\TEMP\jenkins10892010657994167206.bat

C:\ProgramData\Jenkins\.jenkins\workspace\selenium_grid_on_gcp>cd
C:\Users\ROG\OneDrive\Documents\GitHub\SeleniumProject\

C:\Users\ROG\OneDrive\Documents\GitHub\SeleniumProject>python
selenium_grid_test.py
Chrome successfully launched!
Google

C:\Users\ROG\OneDrive\Documents\GitHub\SeleniumProject>exit 0
Finished: SUCCESS
```

Fig. Console Output after building

13. Monitoring VM Instances in GCP

1. **Go to Google Cloud Console: Google Cloud Console**
2. **Navigate to Compute Engine:**
   - **Click on Compute Engine → VM instances.**
3. **Select Your Instance:**
   - **Click on the VM instance running Selenium.**
4. **Open the Monitoring Tab:**

- **View CPU, memory, disk, and network usage graphs.**

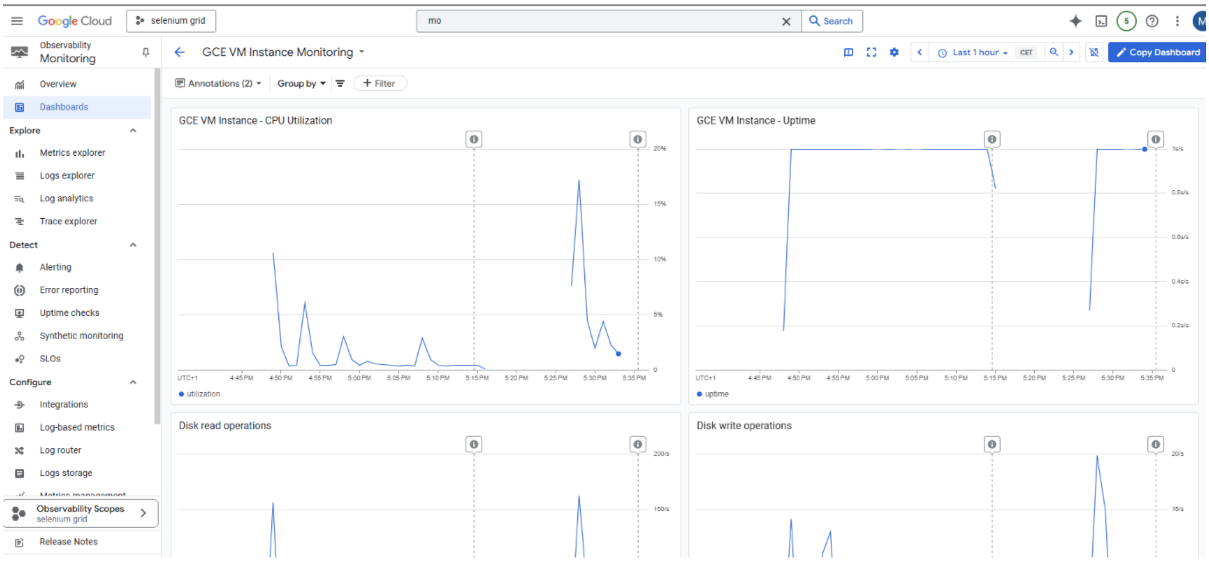


Fig. VM Instance Monitoring

14. Best Practices & Recommendations

- **Which tools, integrations, or plugins enhance continuous testing in the cloud?**
  - Tools such as Selenium Grid, TestNG, PyTest, Allure Reports, and CI/CD tools (Jenkins, GitHub Actions) help enhance testing in cloud environments.
- **How can GCP's built-in logging and monitoring services be leveraged for test management, troubleshooting, and insights?**
  - GCP's Cloud Logging, Cloud Monitoring, and Trace services help track test failures, performance bottlenecks, and resource utilization. Integrating Stackdriver alerts ensures real-time monitoring and proactive issue resolution.

15. Conclusion

By implementing Selenium Grid on GCP VMs using Docker and integrating Jenkins CI/CD, organizations can achieve scalable and cost-effective test automation. This setup ensures optimized performance, security, and continuous delivery of quality software.