## A REPORT

## ON

**Development of an Automated Web Testing Environment Using Python, Docker, and Selenium**

***Submitted by,***

**Mr. L Harshavardhan Naidu - 20211CEI0001**

### *Under the guidance of,*

**Dr. Joe Arun Raja**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER ENGINEERING,**

**(Artificial Intelligence and Machine Learning)**

**At**



**PRESIDENCY UNIVERSITY**

**BENGALURU**

**MAY 2025**

**PRESIDENCY UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that the Internship report **“Development of an Automated Web Testing Environment Using Python, Docker, and Selenium”** being submitted by “**L Harshavardhan Naidu**” bearing roll number “20211CEI0001” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering (AI&ML) is a bonafide work carried out under my supervision.

|  |  |
| --- | --- |
| **Dr. Joe Arun Raja**  Associate Professor  School of CSE&IS  Presidency University | **Dr. Gopal Krishna Shaym**  Professor & HoD  School of CSE&IS  Presidency University |
| **Dr. MYDHILI NAIR**  Associate Dean  School of CSE  Presidency University | **Dr. SAMEERUDDIN KHAN**  Pro-Vice Chancellor - Engineering  Dean -School of CSE&IS  Presidency University |

**PRESIDENCY UNIVERSITY**

**PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

I hereby declare that the work, which is being presented in the report entitled “**Development of an Automated Web Testing Environment Using Python, Docker, and Selenium”** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Engineering Artificial Intelligence and Machine Learning**,, is a record of my own investigations carried under the guidance of **Dr**. **Joe Arun Raja, Associate Professor,** **Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

|  |  |
| --- | --- |
|  | **L Harshavardhan Naidu,**  **20211CEI0001** |

**INTERNSHIP COMPLETION CERTIFICATE**

A close-up of a letter

AI-generated content may be incorrect.

**ABSTRACT**

This report details the internship experience at Innobox Systems Pvt Ltd, centered around Python-based automation of browser testing environments and network performance monitoring. The primary objective of the internship was to develop automation scripts to set up and manage Selenium Grid environments using Docker. A custom Dockerfile was created to install Chrome, Firefox, ChromeDriver, GeckoDriver, and Selenium Server components. Python scripts were used to automatically launch multiple containers and perform SSH operations to initialize browser nodes and the Selenium Hub. In addition, Flent was employed to automate network performance tests, capturing metrics such as latency and bandwidth during test sessions. Postman was also used for validating APIs in controlled network conditions. This hands-on experience enhanced practical knowledge in Python scripting, container-based automation, browser testing, and network performance assessment.

**ACKNOWLEDGEMENTS**

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** Presidency School of Computer Science and Engineering, Presidency University, and **Dr. Gopal Krishna Shaym,** Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr.** **Joe Arun Raja, Associate Professor** School of Computer Science and Engineering, Presidency University for her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the CSE7301 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Project Coordinators SUDHA P and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

**Harshavardhan Naidu**

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Figure Name** | **Caption** | **Page No.** |
| **1.** | Figure 7.1 | Gantt Chart | 13 |
| **2.** | Figure11.1 | Sustainable Development goals | 29 |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT ACKNOWLEDGMENT**  **LIST OF FIGURES** | **v**  **vi**  **vii** |
| **1.** | **INTRODUCTION** | 1 |
|  | 1.1 Background |  |
|  | 1.2 Organization Overview |  |
|  | 1.3 Internship Objectives |  |
| **2.** | **LITERATURE REVIEW** | 2 |
|  | 2.1 Python for Automation |  |
|  | 2.2 Docker and Containerization |  |
|  | 2.3 Selenium Grid |  |
|  | 2.4 Flent for Network Testing |  |
| **3.** | **RESEARCH GAPS OF EXISTING METHODS** | **4** |
|  | 3.1 Complexity of Manual Environment  Setup |  |
|  | 3.2 Incomplete Integration Between Testing  Tools |  |
|  | 3.3 Resource-Heavy Testing Frameworks |  |
|  | 3.4 Absence of Network Sensitivity in Test  Pipelines |  |
|  | 3.5 Limited Remote Control of Containers |  |
|  | 3.6 Scalability and Reusability Challenges |  |
| **4.** | **PROPOSED MOTHODOLOGY** | 6 |
|  | 4.1 System Overview |  |
|  | 4.2 Docker Environment Setup |  |
|  | 4.3 Selenium Grid Deployment via SSH |  |
|  | 4.4 Flent Integration for Network Testing |  |
|  | 4.5 API Testing with Postman |  |
|  | 4.6 Automation Workflow |  |
| **5.** | **OBJECTIVES** | 8 |
|  | 5.1 Main Objectives |  |
|  | 5.2 Learning Objectives |  |
| **6.** | **SYSTEM DESIGN & IMPLEMENTATION** | 10 |
|  | 6.1 System Architecture |  |
|  | 6.2 Tools and Technologies Used |  |
|  | 6.3 Implementation Workflow |  |
| **7.** | **TIMELINE FOR EXECUTION OF PROJECT** | 13 |
| **8.** | **OUTCOMES** | 14 |
|  | 8.1 Technical Outcomes |  |
|  | 8.2 Personal Learning Outcomes |  |
| **9.** | **RESULTS AND DISCUSSIONS** | 16 |
|  | 9.1 Results |  |
|  | 9.2 Discussion |  |
| **10.** | **CONCLUSION** | 18 |
| **11.** | **APPENDICES** |  |
|  | **PSUEDOCODE** | 19 |
|  | **Project work mapping with SDG** | 29 |
|  | **REFERENCES** | 31 |

**Chapter 1**

**INTRODUCTION**

**1.1 Background**

Automation plays a pivotal role in modern software development, especially in testing environments where consistency and speed are critical. Python has emerged as a go-to language for automation due to its simplicity and extensive library support. During my internship at **Innobox Systems Pvt Ltd**, I focused on leveraging Python to automate browser-based testing using **Docker** and **Selenium Grid**. This setup eliminated the need for manual configuration and provided a stable, repeatable testing environment that could be quickly deployed and scaled.

**1.2 Organization Overview**

**Innobox Systems Pvt Ltd** is a technology-driven company specializing in automation solutions and software development services. Their expertise lies in building frameworks that simplify testing, monitoring, and deployment processes. As an intern, my responsibilities included developing Python automation scripts to manage containerized browser environments and integrating tools for network performance testing and API validation.

**1.3 Internship Objectives**

The main objectives of the internship were:

* Automate the deployment of Chrome and Firefox browsers using Docker.
* Use Python scripts to manage and control Docker containers.
* Implement secure SSH-based automation to interact with Selenium Grid services inside containers.
* Integrate **Flent** for network performance testing and **Postman** for validating APIs.
* Gain hands-on experience with scripting, container orchestration, and automated testing tools.

**Chapter 2**

**LITERATURE SURVEY**

Automation tools and testing frameworks have become indispensable in modern development workflows. This section reviews the key technologies used during the internship and their relevance in the industry.

**2.1 Python for Automation**

Python’s readable syntax and vast library ecosystem make it a popular choice for automation tasks. Libraries like **Paramiko** (for SSH access), **subprocess** (for executing system commands), and the built-in **os** module allow for complex scripting capabilities. Python’s flexibility makes it ideal for managing everything from containers to network tools in a unified script.

**2.2 Docker and Containerization**

Docker enables developers to package applications and their dependencies into lightweight, portable containers. This makes deployment fast, consistent, and scalable. In testing environments, using Docker to containerize Selenium nodes ensures a clean, isolated setup that can be reproduced easily across systems. Dockerfiles are used to define the environment, including browsers, drivers, and other tools.

**2.3 Selenium Grid**

Selenium is a powerful framework for automating browser interactions. **Selenium Grid** extends its capabilities by supporting distributed test execution across multiple browsers and systems. Running Selenium Grid within Docker containers allows for parallel execution and simplifies infrastructure management for large-scale automated testing.

**2.4 Flent for Network Testing**

**Flent** (Flexible Network Tester) is a tool used to evaluate network performance by running tests like ping, TCP/UDP transfers, and more. It provides detailed reports and graphical outputs, making it useful for benchmarking how network conditions impact system performance—an important aspect often overlooked in test pipelines.

**2.5 Postman for API Testing**

**Postman** is widely used for testing and validating RESTful APIs. It allows developers to write test scripts in JavaScript and automate checks as part of CI/CD workflows.

**Chapter 3**

**RESEARCH GAPS OF EXISTING METHODS**

Despite the advancements in automation tools, several limitations still hinder the effectiveness of testing systems. This project aimed to address some of these shortcomings through integrated, script-driven solutions.

**3.1 Complexity of Manual Environment Setup**

Setting up browser-based test environments traditionally requires manual installation and configuration of browsers, drivers, and Selenium components. This process is time-consuming and error-prone. Automating the setup using Docker scripts significantly reduces human error and ensures consistency across systems.

**3.2 Incomplete Integration Between Testing Tools**

While individual tools like Docker, Selenium, Flent, and Postman are well-developed, they are often used in isolation. There’s a noticeable lack of frameworks that bring these tools together into a single workflow. This fragmentation makes automation harder to manage and scale, especially in complex testing environments.

**3.3 Resource-Heavy Testing Frameworks**

Running Selenium Grid on physical machines or traditional virtual machines consumes substantial system resources. Docker-based containerization offers a lightweight alternative, but many testers lack experience in automating this setup through scripts. This project demonstrated how Python can simplify and streamline these deployments.

**3.4 Absence of Network Sensitivity in Test Pipelines**

Most test environments assume ideal network conditions, which isn’t reflective of real-world user experiences. Poor latency, packet loss, and other network issues can affect application behavior. Flent provides a way to simulate and measure these effects, yet it's seldom integrated into automated test pipelines. This project addressed this by embedding Flent into the test flow.

**3.5 Limited Remote Control of Containers**

Although Docker containers offer isolated environments, controlling services within them remotely is often a challenge. Without SSH capabilities, it’s difficult to automate actions inside containers dynamically. Using **Python with Paramiko**, this project enabled secure, remote script execution within containers, improving control and flexibility.

**3.6 Scalability and Reusability Challenges**

Many testing setups are tailored to specific projects and are hard to reuse or scale. A flexible, modular automation framework is essential for adapting to different projects or environments. This internship focused on developing such a framework, making it easier to replicate and extend the solution across teams and use cases.

**Chapter 4**

**PROPOSED MOTHODOLOGY**

The methodology developed during this internship focuses on creating a fully automated and scalable test environment for browser-based applications. It leverages Python for scripting, Docker for containerized environments, and Selenium Grid for distributed browser testing. Additionally, network performance is assessed using Flent, and API validations are carried out through Postman. This chapter outlines the structured approach taken to integrate all components into a unified workflow.

**4.1 System Overview**

The proposed system operates as a comprehensive automation framework with the following capabilities:

* Uses a tailored Dockerfile to set up Chrome, Firefox, their respective drivers (ChromeDriver and GeckoDriver), Selenium Server, and SSH within containers.
* Relies on Python automation scripts to build Docker images, launch containers, and control them via SSH.
* Integrates the Flent tool to measure network performance metrics during or prior to test execution.

Incorporates Postman to validate APIs, enabling full-stack testing alongside browser automation.

**4.2 Docker Environment Setup**

A custom Dockerfile forms the foundation of the setup, responsible for:

* Installing essential packages and dependencies for browser execution.
* Deploying Selenium Server components and configuring them for Grid functionality (Hub and Nodes).
* Enabling SSH access for external control and automation.
* Including startup scripts for automatically initiating Selenium services.

Using this Dockerfile, Python scripts were created to build the image and spin up four containers: one designated as the Selenium Hub and the others as browser nodes (both Chrome and Firefox). Ports and volumes were mapped to allow seamless communication and data access.

**4.3 Selenium Grid Deployment via SSH**

Once the containers were running, SSH access was established using Python’s **Paramiko** library to control them remotely. The script performed the following:

* Started the Selenium Hub in the designated container.
* Initialized browser nodes in the remaining containers.
* Registered each node with the central hub.
* Verified node-hub connectivity to ensure test readiness.

This approach enabled dynamic and hands-free deployment of Selenium Grid services.

**4.4 Flent Integration for Network Testing**

Flent was used to analyze the impact of network conditions on automated testing:

* Python scripts executed a variety of Flent tests such as ping, TCP download/upload, and UDP flooding.
* Results were saved in compressed .gz format for storage efficiency.
* The data was then parsed and visualized using **matplotlib** to produce graphs on latency, jitter, and throughput.
* These insights helped evaluate how network conditions could influence the accuracy and reliability of test results.

**4.5 API Testing with Postman**

While the core focus was on browser automation, the environment also supported API testing through Postman. Backend services were tested under simulated network conditions using Flent, providing a clearer understanding of how varying connectivity could affect API responsiveness and performance.

**4.6 Automation Workflow**

The automation process was designed to require minimal user intervention. The high-level workflow included:

1. Building Docker images.
2. Launching containers automatically.
3. Establishing SSH sessions to containers.
4. Deploying Selenium Hub and browser nodes.
5. Running Flent tests and generating performance reports.
6. Performing browser-based and API testing.

This end-to-end automation resulted in a faster, more consistent, and network-aware test environment.

**Chapter 5**

**OBJECTIVES**

The internship aimed to design a fully automated, scalable browser testing environment using Python. The goal was to bridge the technical and operational gaps found in conventional test setups by merging browser testing, container orchestration, remote container management, and network performance monitoring.

**5.1 Main Objectives**

1. Create Python scripts to automate the entire lifecycle of Docker containers.
2. Develop a Dockerfile that installs web browsers, drivers, Selenium Server (both Hub and Node), and SSH services.
3. Automate the launch and management of Selenium Grid containers, including port mapping and health checks.
4. Use **Paramiko** to enable SSH-based control over containers for remote Selenium service execution.
5. Configure Selenium Grid to allow dynamic registration of browser nodes with the central hub.
6. Integrate **Flent** for running network performance tests and visualize the results through custom-generated graphs.
7. Enable API validation using **Postman**, ensuring backend reliability under varying network conditions.
8. Ensure the entire system is reusable, modular, and can be replicated or scaled easily.

**5.2 Learning Objectives**

* Gain practical experience in Python scripting and automation.
* Understand and apply Docker principles to real-world use cases.
* Learn the internal workings of Selenium Grid and how to automate its deployment.
* Explore network testing techniques using **Flent** and analyze the impact on test performance.
* Develop problem-solving abilities by integrating multiple tools into a unified, automated pipeline.

**Chapter 6**

**SYSTEM DESIGN & IMPLEMENTATION**

This chapter details the design and technical implementation of the project. The system was constructed as a layered framework to facilitate modular automation while maintaining clarity and ease of scaling.

**6.1 System Architecture**

The framework was organized into three functional layers:

* **Infrastructure Layer**: Based on Docker, this layer hosts all containers for browsers and Selenium components.
* **Automation Layer**: Python handles all logic, including container operations, SSH communication, Selenium Grid management, and Flent execution.
* **Testing Layer**: Enables browser testing via Selenium and API testing via Postman. This layer also incorporates real-time network performance data for context-aware testing.

A host system acts as the controller for initiating the pipeline, storing logs, and managing outputs.

**6.2 Tools and Technologies Used**

* **Python**: Core scripting language for automation tasks.
* **Docker**: Containerization platform for creating isolated test environments.
* **Selenium Grid**: Framework for distributed browser testing.
* **Paramiko**: Library for secure SSH communication and remote control of containers.
* **Flent**: Network performance testing utility.
* **Matplotlib**: Visualization library used for graphing test data.
* **Postman**: Tool for API validation, integrated with the test pipeline.

**6.3 Implementation Workflow**

**Step 1: Dockerfile Creation**  
A customized Dockerfile was built with:

* Ubuntu as the base OS
* Chrome and Firefox browsers
* ChromeDriver and GeckoDriver
* Selenium Standalone Server
* OpenSSH for remote access  
  This image formed the foundation for all containers.

**Step 2: Python Script for Container Deployment**  
A Python script automated the following:

* Building the Docker image
* Launching one container as the Selenium Hub and three containers as browser nodes
* Assigning appropriate ports and validating container status

**Step 3: SSH-based Selenium Grid Initialization**  
Using Paramiko, the script:

* Connected to each container over SSH
* Started the Selenium Hub in its designated container
* Launched browser nodes and linked them to the hub
* Verified successful registration and status reporting of nodes

**Step 4: Flent Integration**  
A separate flent.py script allowed:

* Execution of network performance tests
* Saving of output in compressed formats
* Plotting of results to visualize factors like latency and jitter  
  This helped identify how unstable network conditions could influence automated test performance.

**Step 5: API Testing with Postman**  
In tandem with browser and network tests, Postman was used to test API responses. These tests were run under varying network conditions, simulating real-world use cases and ensuring backend reliability.

**Chapter-7**

**TIMELINE FOR EXECUTION OF PROJECT**

A graph with blue rectangles

AI-generated content may be incorrect.

Figure 7.1 Gantt Chart

**Chapter 8**

**OUTCOMES**

The internship at **Innobox Systems Pvt. Ltd.** turned out to be a highly valuable and hands-on learning experience. It gave me the chance to apply what I had learned in academics to solve real-world problems, especially in the area of automation and testing infrastructure. By the end of the internship, I had gained not only technical expertise but also practical insights into building scalable, automated systems.

**8.1 Technical Outcomes**

1. **Python Automation:**  
   I wrote Python scripts that automated the entire testing environment setup—from building Docker containers to controlling them remotely and deploying Selenium Grid. This helped remove manual steps from the process and made testing faster and more reliable.
2. **Custom Docker Setup:**  
   I created a Dockerfile that packaged everything—browsers (Chrome, Firefox), Selenium Server, drivers, and SSH—into a single image. This made it easy to spin up test containers consistently and quickly.
3. **Selenium Grid Control via SSH:**  
   Using the Paramiko library, I connected to each container over SSH and launched the Selenium Grid services. This allowed for dynamic and remote test setup, which would otherwise require manual steps.
4. **Network Testing with Flent:**  
   I integrated Flent to measure things like latency and bandwidth. The results were saved automatically, and I used matplotlib to generate graphs that helped me understand how network conditions might affect test performance.
5. **API Testing with Postman:**  
   While working on browser automation, I also used Postman to run API tests. Running these tests under different network conditions helped check how stable the backend was.
6. **Fully Automated Workflow:**  
   I managed to tie everything together into a single workflow that could be run with minimal input. The entire setup—from container creation to test execution—was automated end to end.

**8.2 Personal Learning Outcomes**

* I got much more comfortable working with Linux and Docker, which were both completely new to me at the start.
* My Python skills improved a lot, especially when dealing with real-time issues and debugging.
* I learned how to plan and manage my time better, especially when juggling multiple tools and technologies.
* One of the biggest takeaways was the confidence I gained in building something practical from scratch and making it work in the real world.

**Chapter 9**

**RESULTS AND DISCUSSIONS**

The main goal of this project was to build an automated testing system using Python and Docker. By the end, I had built a working solution that was fast, repeatable, and fairly easy to manage. Here’s a summary of what was achieved and a few things that could be improved.

**9.1 Results**

**Docker-Based Setup:**

* I successfully created and launched four containers automatically.
* Each one was preloaded with the necessary tools and could run Chrome or Firefox tests.
* The Selenium Hub and browser nodes connected properly without any manual setup.

**Selenium Grid Automation:**

* The grid worked as expected. It could manage multiple test sessions running in parallel, which sped up the testing process.

**SSH-Based Control:**

* I used Python to log into each container and start Selenium services remotely. This helped simulate a real-world distributed system.

**Network Testing Using Flent:**

* I ran different Flent tests like ping and TCP upload/download.
* The results were saved and visualized using graphs. This helped understand how test results might vary under different network conditions.

**API Testing:**

* Postman was used to test backend APIs while network stress was applied.
* The APIs held up well, showing that the backend was resilient..

**9.2 Discussion**

The integration of all these tools—Docker, Selenium Grid, SSH, and Flent—through Python scripting made the testing framework really flexible and easy to reuse. Automating everything meant I didn’t have to rely on any manual setup, which saved a lot of time and effort.

Adding Flent to the mix turned out to be a great idea. Even though it’s not commonly used with browser automation, it gave a more complete picture of how real-world network conditions could affect test reliability.

**Challenges I faced:**

* Starting all the containers at once was a bit heavy on systems with low RAM.
* Installing and configuring Flent inside containers took some trial and error.
* Monitoring live logs from Selenium could be improved in the future with better tools like centralized dashboards.

Overall, I’m happy with how the system turned out. It achieved the original goals and has room to grow into something even more powerful.

**Chapter 10**

**CONCLUSION**

This internship gave me a solid chance to apply my Python automation skills in a real-world project. The main objective was to build an automated, container-based browser testing environment, and I’m proud to say that was successfully accomplished.

I developed a complete workflow that:

* Builds Docker containers with browsers and Selenium Grid.
* Connects to those containers via SSH.
* Sets up Selenium Grid dynamically.
* Runs network tests with Flent.
* Supports backend API testing with Postman.

Each of these parts came together to form an automated, flexible testing system. Python played a central role in tying everything together, and Docker made it easy to replicate the environment consistently. SSH scripting gave me fine control over each part of the setup, and Flent added value by showing how network conditions can affect tests.

**In summary, this internship helped me:**

* Get better at writing real-world Python automation scripts.
* Learn how to use Docker in a practical setting.
* Understand how to manage and automate testing infrastructure.
* Appreciate the importance of performance testing in automation workflows.

This project was not just about building a tool—it was about learning how different technologies can work together in a real system. It has definitely boosted my confidence and prepared me for more advanced projects in automation and infrastructure management.

**PSUEDOCODE**

**Flent**import subprocess

import gzip

import json

import os

import glob

import matplotlib.pyplot as plt

from datetime import datetime

# === Create a structured directory to store test results ===

# Main folder to hold all Flent test results

BASE\_DIR = "flent\_results"

os.makedirs(BASE\_DIR, exist\_ok=True)

# Subfolder by current date

date\_folder = datetime.now().strftime("%d-%m-%Y")

date\_path = os.path.join(BASE\_DIR, date\_folder)

os.makedirs(date\_path, exist\_ok=True)

# Time-based folder for this test session

time\_folder = datetime.now().strftime("%H-%M-%S")

session\_path = os.path.join(date\_path, time\_folder)

os.makedirs(session\_path, exist\_ok=True)

# === Display test options to user ===

print("Select a network test to run:")

print("1. TCP Download")

print("2. TCP Upload")

print("3. Bursts")

print("4. Ping")

print("5. UDP Flood")

print("6. Run All Tests")

choice = input("Enter your choice (1-6): ").strip()

# IP of the host to test against

target\_ip = "10.10.40.208"

saved\_files = []

# Mapping user choice to Flent test names

flent\_tests = {

"1": "tcp\_download",

"2": "tcp\_upload",

"3": "bursts",

"4": "ping",

"5": "udp\_flood"

}

def find\_latest\_flent\_file():

"""Find the most recently created Flent .gz file in current directory."""

files = glob.glob("\*.flent.gz")

if not files:

return None

return max(files, key=os.path.getmtime)

def run\_flent\_test(test\_name):

"""Run a Flent test and move the result to the session folder."""

print(f"\nRunning Flent test: {test\_name}...")

command = ["flent", test\_name, "-l", "60", "-H", target\_ip]

try:

result = subprocess.run(command, check=True, capture\_output=True, text=True)

print("Test completed successfully.")

print(result.stdout)

latest\_file = find\_latest\_flent\_file()

if latest\_file:

destination = os.path.join(session\_path, os.path.basename(latest\_file))

os.rename(latest\_file, destination)

saved\_files.append(destination)

else:

print(f" No Flent output file found for {test\_name}.")

except subprocess.CalledProcessError as e:

print(f"Error during {test\_name} test:\n{e.stderr}")

# === Run selected tests ===

if choice in flent\_tests:

run\_flent\_test(flent\_tests[choice])

elif choice == "6":

for test in flent\_tests.values():

run\_flent\_test(test)

else:

print("Invalid input. Please enter a number between 1 and 6.")

exit()

# === Plotting Results ===

def extract\_and\_plot(file\_path):

"""Extracts metrics from Flent .gz file and generates a line graph."""

print(f"\nProcessing: {os.path.basename(file\_path)}")

try:

with gzip.open(file\_path, 'rt', encoding='utf-8') as f:

data = json.load(f)

x\_vals = data.get('x\_values', [])

metric\_name = 'Ping (ms) ICMP'

y\_vals = data.get('results', {}).get(metric\_name, [])

if not x\_vals or not y\_vals:

print (f"No data found for plotting in {file\_path}")

return

# Create the plot

plt.figure(figsize=(10, 5))

plt.plot(x\_vals, y\_vals, label=metric\_name, color='blue', linestyle='-', marker='o')

plt.title(f"Flent Test - {os.path.basename(file\_path)}")

plt.xlabel("Time (s)")

plt.ylabel("Latency (ms)")

plt.grid(True)

plt.legend()

# Save the graph

graph\_name = os.path.splitext(os.path.basename(file\_path))[0] + "\_graph.png"

graph\_path = os.path.join(session\_path, graph\_name)

plt.savefig(graph\_path)

plt.close()

print (f" Graph saved to: {graph\_path}")

except Exception as e:

print (f"Failed to process {file\_path}: {e}")

# === Process all saved test results ===

for result\_file in saved\_files:

extract\_and\_plot(result\_file)

print (f"\n All test results and graphs are saved in:\n{session\_path}")

print ("Network testing completed successfully.")

**Dockerfile**

# Use Ubuntu 22.04 as the base image

FROM ubuntu:22.04

# Avoid interactive prompts during package installations

ENV DEBIAN\_FRONTEND=noninteractive

# Install required packages and dependencies

RUN apt-get update && apt-get install -y --no-install-recommends \

ca-certificates curl dnsutils iproute2 iputils-ping net-tools \

openssh-server openssh-client python3 python3-pip python3-venv \

sudo wget iptables jq unzip bzip2 xz-utils pciutils lsb-release \

fonts-liberation xdg-utils openjdk-11-jre netcat \

libdbus-glib-1-2 libxt6 libxcb-shm0 libxcb-dri3-0 libgbm1 \

libgtk-3-0 libnss3 libxss1 libasound2 libatk-bridge2.0-0 \

libcups2 libxrandr2 libappindicator3-1 libdrm2 libx11-xcb1 libpci3 \

&& rm -rf /var/lib/apt/lists/\*

# Set environment variables for Java

ENV JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

ENV PATH="$JAVA\_HOME/bin:$PATH"

# Create a user for Selenium and allow passwordless sudo

RUN useradd -m -s /bin/bash selenium\_grid && \

echo "selenium\_grid:kernel" | chpasswd && \

usermod -aG sudo selenium\_grid && \

echo "selenium\_grid ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers

# Configure SSH server

RUN mkdir -p /var/run/sshd && \

sed -i 's/#PasswordAuthentication yes/PasswordAuthentication yes/' /etc/ssh/sshd\_config && \

sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin no/' /etc/ssh/sshd\_config && \

sed -i 's/#Port 22/Port 22/' /etc/ssh/sshd\_config

# Set working directory

WORKDIR /home/selenium\_grid

# Install Chrome, ChromeDriver, Firefox, and Geckodriver

RUN mkdir -p chrome chromedriver firefox && \

wget -q -O /tmp/chrome.zip "https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.204/linux64/chrome-linux64.zip" && \

unzip -q /tmp/chrome.zip -d chrome && \

chmod +x chrome/chrome && \

wget -q -O /tmp/chromedriver.zip "https://storage.googleapis.com/chrome-for-testing-public/131.0.6778.204/linux64/chromedriver-linux64.zip" && \

unzip -q /tmp/chromedriver.zip -d chromedriver && \

chmod +x chromedriver/chromedriver && \

wget -q -O /tmp/firefox.tar.bz2 "https://ftp.mozilla.org/pub/firefox/releases/134.0b9/linux-x86\_64/en-US/firefox-134.0b9.tar.bz2" && \

tar -xjf /tmp/firefox.tar.bz2 -C firefox --strip-components=1 && \

chmod +x firefox/firefox && \

wget -q -O /tmp/geckodriver.tar.gz "https://github.com/mozilla/geckodriver/releases/download/v0.35.0/geckodriver-v0.35.0-linux64.tar.gz" && \

tar -xzf /tmp/geckodriver.tar.gz -C /home/selenium\_grid && \

chmod +x /home/selenium\_grid/geckodriver && \

rm -rf /tmp/\*

# Download Selenium Server JAR

RUN wget https://github.com/SeleniumHQ/selenium/releases/download/selenium-4.27.0/selenium-server-4.27.0.jar -O selenium-server.jar

# Setup Python environment and install libraries

RUN python3 -m venv venv && \

. venv/bin/activate && \

pip install --upgrade pip && \

pip install selenium requests webdriver-manager && \

pip freeze > requirements.txt

# Set environment paths

ENV PATH="/home/selenium\_grid/venv/bin:/home/selenium\_grid:/home/selenium\_grid/chromedriver:/home/selenium\_grid/firefox:/home/selenium\_grid/chrome:$PATH"

ENV CHROME\_BIN=/home/selenium\_grid/chrome/chrome

ENV FIREFOX\_BIN=/home/selenium\_grid/firefox/firefox

ENV GECKODRIVER\_PATH=/home/selenium\_grid/geckodriver

# Change ownership to selenium\_grid user

RUN chown -R selenium\_grid:selenium\_grid /home/selenium\_grid

# Expose necessary ports

EXPOSE 22 4444

# Switch to non-root user

USER selenium\_grid

# Default command

CMD ["/bin/bash"]

**Selenium\_grid\_server**

import subprocess

import json

import time

import os

# Load container setup details

with open("selenium\_server\_details.json", "r") as f:

config = json.load(f)

build\_command = config["command\_a"]

run\_command\_template = config["command\_b"]

ssh\_config = config["ssh"]

containers = config["containers"]

terminal\_processes = []

def stop\_and\_remove\_container(name):

subprocess.run(f"docker rm -f {name}", shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

def build\_and\_launch\_container(container):

name = container["name"]

ssh\_port = container["ssh\_port"]

selenium\_port = container["selenium\_port"]

stop\_and\_remove\_container(name)

print(f"\n Building container: {name}")

result = subprocess.run(build\_command, shell=True, text=True)

if result.returncode == 0:

print("Build successful.")

else:

print("Build failed.")

return

print(f" Starting container: {name}")

run\_cmd = run\_command\_template.format(container\_name=name, ssh\_port=ssh\_port, selenium\_port=selenium\_port)

subprocess.run(run\_cmd, shell=True)

time.sleep(3)

subprocess.run(f"docker exec {name} sudo service ssh start", shell=True)

print(f"SSH service started in {name}")

def prepare\_ssh\_start\_commands():

commands = []

for container in containers:

ssh\_cmd = (

f"sshpass -p {ssh\_config['password']} ssh -o StrictHostKeyChecking=no "

f"{ssh\_config['username']}@{ssh\_config['hostname']} -p {container['ssh\_port']} "

f"'java -jar selenium-server.jar standalone --host 0.0.0.0 --port {container['selenium\_port']}'"

)

commands.append(ssh\_cmd)

return commands

def open\_terminals(commands):

for cmd in commands:

print(f"Launching terminal for: {cmd}")

process = subprocess.Popen(f'gnome-terminal -- bash -c "{cmd}; exec bash"', shell=True)

terminal\_processes.append(process)

def cleanup():

print("\n Cleaning up terminals and containers...")

for process in terminal\_processes:

process.terminate()

try:

process.wait(timeout=5)

except subprocess.TimeoutExpired:

process.kill()

for container in containers:

name = container["name"]

subprocess.run(f"docker exec {name} pkill -f selenium-server", shell=True)

subprocess.run(f"docker stop {name}", shell=True)

subprocess.run(f"docker rm -f {name}", shell=True)

subprocess.run("sudo pkill -f docker-pr", shell=True)

def main():

try:

for container in containers:

build\_and\_launch\_container(container)

time.sleep(2)

ssh\_commands = prepare\_ssh\_start\_commands()

open\_terminals(ssh\_commands)

while True:

time.sleep(1)

except KeyboardInterrupt:

print("\nScript interrupted by user.")

cleanup()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**json.file**

{

"command\_a": "docker build -t selenium-h .",

"command\_b": "docker run -dit -p {ssh\_port}:22 -p {selenium\_port}:{selenium\_port} --name {container\_name} selenium-h",

"ssh": {

"hostname": "localhost",

"username": "selenium\_grid",

"password": "kernel",

"timeout": 10

},

"java\_command\_template": "java -jar selenium-server.jar standalone --host 0.0.0.0 --port {selenium\_port}",

"containers": [

{

"name": "selenium-h1",

"ssh\_port": 2234,

"selenium\_port": 4445

},

{

"name": "selenium-h2",

"ssh\_port": 2235,

"selenium\_port": 4446

},

{

"name": "selenium-h3",

"ssh\_port": 2236,

"selenium\_port": 4447

},

{

"name": "selenium-h4",

"ssh\_port": 2237,

"selenium\_port": 4448

}

]

}

**Project work mapping with SDG**

A chart of goals for a sustainable development

AI-generated content may be incorrect.

Figure11.1: Sustainable Development goals

This project centers around streamlining and improving software testing systems through the use of Python, Docker, and Selenium Grid. By automating time-consuming tasks and enabling scalable container-based setups, it supports the development of modern, efficient infrastructure. The goal is to replace traditional manual testing with sustainable, automated processes that are better suited for today’s fast-paced tech landscape. In doing so, the project directly supports innovation and the creation of smarter, more resilient digital systems.

**SDG 4: Quality Education**

As part of a hands-on internship, this project offers practical experience in automation, scripting, and systems design. Beyond just writing code, it provides exposure to real-world challenges and solutions in software infrastructure. This kind of experiential learning helps bridge the gap between academic study and professional expectations, empowering learners with the skills needed to thrive in technical careers. It promotes continuous, lifelong learning by giving meaningful context to theoretical knowledge.

**SDG 12: Responsible Consumption and Production**

By automating browser-based testing and deploying lightweight containers, the project encourages smarter use of system resources. Instead of repeatedly setting up heavy testing environments, it reuses and optimizes containers, reducing energy usage and hardware strain. This not only enhances efficiency but also supports more sustainable software development practices—minimizing waste and improving productivity with fewer computing resources.

**SDG 8: Decent Work and Economic Growth**

Through automation, the project helps reduce the need for repetitive manual testing tasks, freeing up developers and QA engineers to focus on more creative and impactful work. It reflects a shift toward smarter workflows where scripting and tool integration enhance overall productivity. By adopting these modern practices, the project supports a forward-thinking work culture that values efficiency, innovation, and professional growth.

**REFERENCES**

**SeleniumHQ**. (2025). *Selenium Documentation*. Retrieved from <https://www.selenium.dev/documentatio>n

**Docker Inc**. (2025). *Docker Documentation*. Retrieved from

https://docs.docker.com

**Flent Project.** (2025). *Flent – The Flexible Network Tester*. Retrieved from <https://flent.org>

**Paramiko Developers**. (2025). *Paramiko – SSH Library for Python*. Retrieved from https://docs.paramiko.org

**Postman. (2025).** *Postman API Platform*. Retrieved from <https://www.postman.com>

Python Software Foundation. (2025). *Python 3.12 Documentation*. Retrieved from <https://docs.python.org/3>

**Matplotlib Developers**. (2025). *Matplotlib: Visualization with Python*. Retrieved from <https://matplotlib.org>

**GitHub. (2025).** *Docker-Selenium Example Projects*. Retrieved from <https://github.com/SeleniumHQ/docker-selenium>