Rion Nakayama

AUTOMATION TESTING WITH SELENIUM GRID IN THE VIRTUAL LAB ENVIRONMENT

Bachelor's thesis

Bachelor of Engineering

Information Technology

2023

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Rion Nakayama
Thesis title	Automation testing with Selenium Grid in the Virtual Lab environment
Commissioned by	Xamk QA workshop project
Year	2023
Pages	40 pages
Supervisor(s)	Heikki Brotkin

ABSTRACT

UI testing plays an important role in the process of developing web applications as user interface is the foremost part that affects the user experience. Ensuring compatibility of web applications across different platforms and browsers is one challenge. The availability of a variety of browsers means that web applications can face a variety of problems. Selenium Grid is a convenient tool for performing automated cross browser and cross platform testing.

The objective of the study was to set up an environment for automated UI testing with Selenium Grid in the Xamk Virtual Lab environment and to verify the effectiveness of the combination. Xamk Virtual Laboratory is suitable for working with multiple virtual machines. Therefore, it was expected that Selenium Grid environment could be developed and maintained efficiently.

The test environment for running Selenium Grid was created in the Xamk Virtual Lab, for running automated UI tests on browser application with Windows 10, Windows 11, Linux and Android. Several tests were conducted using MediaWiki web page, to validate the usefulness of the setup.

The combination of Selenium Grid and Virtual Laboratory worked well. It made it easy to create and manage a Grid environment, with a user-friendly interface. However, there is room for improvement as the current implementation lacks stability, leading to occasional flaky tests.

Keywords: Selenium Grid, Automation testing, Virtualization, Appium

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In numerous components of web applications, a user interface would be the biggest and most important element for general users. As the term "User Interface" suggests, the UI represents the foremost part with which users engage. Quality of the user interface directly affects users' impressions of the application. It is essential that the UI functions smoothly to retain and attract customers. Therefore, UI testing plays an important role in the software development process. UI testing is conducted to measure the performance and overall functionality of the visual elements of an application. It will confirm that the application does not have unexpected results or bugs by inspecting visual elements and functions of them.

One of the biggest difficulties in testing on a web application is ensuring compatibility across different browsers, browser versions, and platforms. The variety of a combination of browsers and platforms means web applications can face diverse problems on each configuration. Since testing all functions and use case is very time-consuming, an automation test is used often. Selenium is one of the solutions for UI testing automation. Since it enables to run the automation test script on different machines and different browsers, it makes it easy to verify compatibility. One can more efficiently evaluate how well a web application works in different environments.

Implementing Selenium Grid environment on Xamk Virtual Laboratory was my duty during my practical training at Xamk QA workshop project. The project was established for the purpose of promoting a new collaboration approach between educational institutions and software companies, and develop QA competences of the IT sector in South-Savo area. In this project one of the main focuses was Xamk Virtual Laboratory. It is a virtual environment created and maintained by Xamk and widely used for educational purposes. During the project it was attempted to utilize this Virtual Laboratory for collaboration between companies and students.

1.1 Xamk QA workshop

The Xamk Software Quality Assurance Workshop was established in September 2021 and concluded in August 2023. The aim of the project was to promote cooperation between educational institution and local software companies and to boost the vitality of the software industry in the South-Savo area.

One of the main focuses of the project was developing new collaborating models of Xamk and IT companies in the South-Savo area to exchange QA and software development related knowledge, which led to organising meeting events for software developers called "guild meeting". These meetings were held regularly starting in June 2022 and were attended by many IT workers, including engineers from local companies, Xamk staff, and project workers. In the meetings, participants got to know each other, shared ideas and discussed a decided topic.

Another key focus was organizing the workshops where students and companies could work on the QA assignments together. During the project, lots of collaboration with local companies occurred on a wide range of topics, including penetration testing, load testing, and user interface testing. The Xamk Virtual Laboratory was actively used in formulating solutions for the assignments. (Jantunen, 2023.)

1.2 Motivation of the study

For testing compatibility across different platforms and browsers, Selenium Grid is a convenient tool. However, there are some difficulties as well, since multiple physical devices or virtual machines have to be prepared to create the testing environment. Utilizing actual computers would be costly and complex to maintain, because of the requirement of multiple machines with various operating systems. Deploying in a cloud environment like AWS or Azure would also cost quite a lot. Another challenge is the scalability of the test scope. It would not be very flexible to scale up or down the test environment with implementations on hardware resources or locally installed virtual machines.

As Xamk Virtual Lab environment is suitable for working with multiple virtual machines, it is considered that conducting Selenium Grid test on Virtual Laboratory would be a good combination. It was expected that Virtual Laboratory allows to create and maintain the Selenium Grid setup efficiently, and facilitate the operation of UI testing with a variety of configurations.

The aim of this thesis is to implement a Selenium Grid environment on the Xamk Virtual environment and conduct automation tests there, to examine how well the combination works and what impact it has on the test execution.

2 SELENIUM

Selenium is a suite of web browser automation tools broadly used by software developers and QA engineers. It was originally developed for automation testing purposes, but can also be used for web scraping or automation of recurring tasks on web applications. The core feature of Selenium is remote control of web browser instances and imitation of user interaction with web applications, such as visiting URL, inserting text into textbox, clicking buttons, or selecting a check box.

Selenium was initially started by Jason Huggins of ThoughtWorks in 2004, for the purpose of testing an in-house Time and Expenses application. He developed a JavaScript program called "JavaScriptTestRunner", which later became "Selenium-Core". Many workers at ThoughtWorks got excited about the potential to develop into a reusable testing framework for other web applications, and the project was open-sourced the same year. (Software Freedom Conservancy, n.d.a.)

Selenium RC (Remote Control) had been the main part of the Selenium project for a long time, in the first version of Selenium. It consisted of two elements; Selenium server and client libraries. Selenium server acts as an HTTP proxy, which intercepts HTTP requests and responses exchanged between the browser and web application. Client libraries offer programming support for each language as an interface between the automation commands and Selenium server. When the test command is executed, the client library communicates with the server,

which passes each Selenium command to the browser. (Software Freedom Conservancy, n.d.b.)

With the release of Selenium 2.0, WebDriver became the main tool of Selenium project as the successor of Selenium RC. Whereas Selenium RC was using server for injecting JavaScript program in browser, WebDriver used the native driver concept to communicate with the web application. Hence the need for a proxy server has been eliminated and made test speed faster, since it interacts directly with browser. The API has also been improved to be more object-oriented and easier to understand. (Software Freedom Conservancy, n.d.b.)

In Selenium 3, the Selenium RC code has been completely removed from the implementation. In addition, Selenium 3 has been approved as a standard by the World Wide Web Consortium (W3C). This has led to major browser vendors becoming more active in supporting the WebDriver specification and providing the necessary functionality along with the browser. Also at this time, the mobile testing project "Appium" was launched. (Gundecha & Avasarala, 2018.)

In Selenium 4, the most important upgrade is the removal of the JSON Wire Protocol. Previously, JSON Wire Protocol has been used for transferring data between server and client, over HTTP. Now in Selenium 4, the client and server now communicate directly with each other via the W3C protocol. It has three main tools: WebDriver, IDE and Grid.

2.1 Selenium WebDriver

Selenium WebDriver is an API and protocol which supports a standardized way to examine and manipulate browser sessions. It offers a language-neutral interface, allowing users to choose their preferred programming languages to write scripts. It can drive the browser either locally or remotely. (Stewart & Burns, 2018.) WebDriver sessions use WebDriver API, client library and browser driver, plus other frameworks as required. WebDriver API is the set of commands for exploring and controlling DOM elements of web applications, and replicating user behaviour. Client libraries are specific for each programming language. These

are required to interact with WebDriver API. The browser driver behaves as a mediator and it is responsible for controlling the actual browser. If test commands are executed, WebDriver forwards them to the driver, and the driver translates them into actions on the browser. Each browser requires its own specific driver. In most cases, those are developed by the browser vendor itself. (Chaubal, 2018.)

2.2 Selenium Grid

Selenium Grid is one of tools of Selenium suite that is focusing on cross platform testing. It allows execution of test scripts on multiple machines, which makes it easy to run test cases on different combinations of browsers, browser versions and operating systems. It is also possible to run tests on several machines in parallel, thus enabling a reduction of testing time.

Grid is composed of the following six elements (shown in Figure 1):

- Router
- Distributor
- Session Map
- Session Queue
- Node
- Event Bus

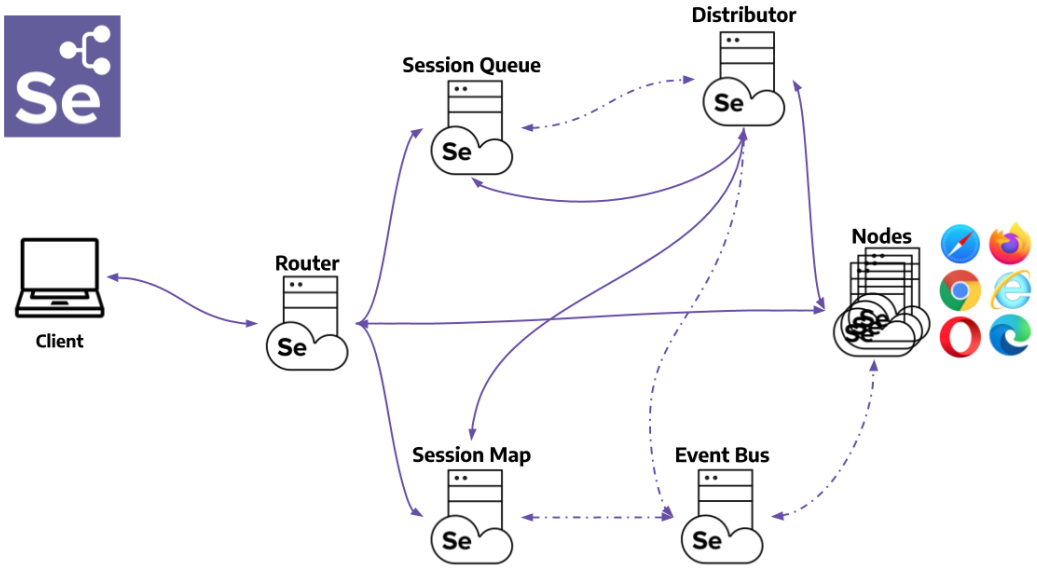

Figure 1. Grid components (Software Freedom Conservancy, n.d.b.)

Router works as a gateway of Grid, which receives all traffic from outside and deliver them to correct place. When the router receives a session request, it will be sent to Session Queue. It also acts as a load balancer to observe the circumstances of each component.

Distributor is used for registering and managing all Nodes and their information. Once Node machine is started, it will send a registration event over Event Bus, in order to register with the Distributor. When a request is delivered, Distributor attempts to verify its existence by reaching the Node via HTTP. Upon a successful request, the Distributor enrolls the Node and monitors its capabilities.

Session Map preserves the mapping between the session ID and the corresponding Node tasked with executing the session. It helps the router obtain information about the currently running session and assigns traffic to the node appropriately.

Session Queue retains incoming session requests in a first-in, first-out (FIFO) sequence. The router forwards new requests to the Session Queue. If the requested capabilities match information of an available Node, Distributor tries to forward the session to the Node slot. If the request is not executed within the expiration time and timeout, the request will be deleted.

Nodes are responsible for executing test scripts on the machine itself. A Grid can have one or more Nodes with different Operating Systems. Nodes register with the Distributor by sending registration messages containing their configuration data. Node automatically registers all available browser drivers on the path of the machine where it is running, by default. For Firefox and Chromium-based browsers, one slot is created for each available CPU, while only one slot is created for Safari.

Event Bus functions as a communication channel between Session Queue, Session Map, Distributor and Nodes, since Grid avoids using expensive HTTP

calls and communication between internal components is mostly done through messages. (Software Freedom Conservancy, n.d.b.)

2.2.1 Roles

Since Grid has six components, there are several options on how to configure them. The developers can choose a suitable role depending on their needs. There are standalone mode, Hub-Node mode, and Distributed mode.

With the standalone mode one can start full function of Selenium Grid at once on a single computer. It runs all of the six components on one machine with one command. This mode usually used for the purpose of developing and debugging tests by using RemoteWebDriver locally, for quickly running test suites before pushing code, or for having easy Grid setup with CI/CD tools.

Hub-Node is the most used function because of its powerful and handy usage. It allows you to control different machines in a single grid and have only one entry point to run test cases in multiple environments. In this mode, Hub machine takes responsibility for five components except Node (Router, Distributor, Session Map, Session Queue, and Event Bus). Hub and Nodes communicate with each other using HTTP and Event Bus. To initiate the registration process, Node passes a message to the Hub over Event Bus. When the Hub gets the message, it uses HTTP to verify its existence. In order to register a Node to the Hub, it is required to expose Event Bus port on both Node and Hub machines, which is 4442 and 4443 by default. This allows both the Hub and Node to communicate.

With the distributed mode, each component will be started individually, and preferably on different machines. In this mode, each component must be started on a specific port number, making startup and management relatively complex. However, this mode is effective when building large grid environments with more than 100 nodes. (Software Freedom Conservancy, n.d.b.)

2.2.2 Remote WebDriver

Remote WebDriver supports Selenium Grid to automate browsers installed on remote computers. It is a class that is implemented under selenium.webdriver class. When using Remote WebDriver, the computer that has the driver and browser is called remote computer or end-node, and the computer that runs the testing code is called client. RemoteWebDriver class is required for directing Selenium tests to the remote computers. By passing the URL and the port number of the grid to the client by using Remote WebDriver, it can connect to the remote computer and forward Selenium tests. When using Remote WebDriver, information is needed on both where to forward commands and which browser to open on the remote machine. (Software Freedom Conservancy, n.d.b.)

2.3 Appium

During the project it was suggested to broaden the test scope to mobile applications and the possibility was explored. It turned out that Appium is the suitable solution for this. Selenium Grid alone does not support mobile testing, but Android devices can be added to the Grid scenario as a node by using Appium.

Appium is an UI automation framework based on Selenium API which is built to facilitate automation testing of any app platform. It supports a wide range of application platforms, including mobile, browser, TV, desktop and more. Appium was initially started by Dan Cuellar who was working as the Test Manager at Zoosk, to automate testing of iOS applications. He implemented a test framework for iOS automation using Selenium syntax. Later, the project was published as open-source and grew into a cross-platform automation tool. (Appium Documentation, n.d.)

Appium is a web server written in node.js, which has a client server architecture. It uses a tool called Appium driver and this assists Appium to automate and communicate with various different kinds of platforms. Driver is a kind of pluggable software module and it maps the protocol to automate the behaviour

which is requested through an Appium server. It is responsible for implementing the internal Appium interface that represents the WebDriver protocol. There are specific drivers depending on the platform and how it implements the protocol varies by driver. XCUITest is used for iOS apps and UiAutomator2 is used for Android, for example.

Because of the relay feature of Selenium Grid, it is possible to connect Appium instances to Selenium Grid as nodes. This feature allows Selenium Grid to connect to external services that support WebDriver, such as Appium and cloud providers, by relaying commands to the service endpoints. (Software Freedom Conservancy, n.d.b.)

3 VIRTUALIZATION

Virtualization is a way of utilizing physical computer hardware more efficiently by using virtual machines. By creating an abstraction layer, hardware resources of a single computer can be divided into several virtual machines and each of those operates as an independent computer. The computer that provides physical resources is called host and the virtual workload is called guest. (IBM, n.d.) Any hardware resources can be virtualized including processors, storage, memory and network connectivity. By the means of virtualization, one can run many different operating system workloads on one computer at the same time. (Portnoy, 2016.)

Virtualization is done through a software called hypervisor. This is also called virtual machine monitor (VMM) and this is responsible for creating and managing virtual machines. There are two types of hypervisors, which is Type1 (bare-metal) and Type2 (hosted).

Type1 is directly installed on top of the hardware, on the same layer of the operating system. Therefore, it can straightly interact with the hardware beneath it and negotiate for allocating resources to virtual workloads, which makes the process more efficient and offers better performance. However, the operation tends to be relatively complex and some level of advanced knowledge is

required. (vmware, n.d.) Type1 is typically used in situations where workloads are resource-intensive, large, or fixed-use, such as data centers, web servers, or cloud computing environments. Examples of Type 1 are KVM and VMware ESXi. (AWS, n.d.)

Type2 is installed on top of the operating system same as other applications, and communicates with the hardware through the OS of the host. The performance is relatively slow and less efficient compared to type1, because it has to interact with the operating system to gain resources for virtual workloads and can use only what OS provided. On the other hand, it is easy to install and handle, as it operates as other normal applications. (vmware, n.d.) Type2 tends to be chosen for personal use or small-scale situations where cost-effectiveness, convenience and portability are valued, such as development environment or desktop workstation. Also, it is preferred in the situation where the user wants to run several OSes simultaneously but has access to only one machine. Examples of Type2 are Oracle VM VirtualBox, VMware Workstation. (AWS, n.d.)

3.1 Oracle VM VirtualBox

The main purpose of using VirtualBox was to create virtual machines with different operating systems and to test the connection of Selenium Grid by running two or three virtual machines locally on the laptop.

VirtualBox is an open-source virtualization software provided by Oracle, and it is a Type2 hypervisor. VirtualBox has been a popular choice for a long time for both enterprise and personal use, because of the simpleness of operation, flexibility and low cost. It is a cross-platform application, which means it can be used on any Operating System. Since it was designed for use on a variety of hardware, it is lightweight and simple to install and work with. Regardless of the simple and lite design, it offers powerful performance with flexible features, including easiness of exporting/importing VM images, compatibility across different platforms, ability to take snapshots, and support for comprehensive hardware. (Oracle, n.d.)

Even though VirtualBox is useful for handling multiple VMs, there is a limit depending on the hardware resources of the host machine. Especially Windows machines require relatively high amounts of memory and storage. To work with a larger number of machines, it is required to move to a cloud-based virtualization platform.

3.2 Xamk Virtual Laboratory

Virtual Laboratory is a virtual environment open for Xamk students, and it is widely used for leaning purposes such as cyber security training or networking exercises. Users can create or access lab instances where they can handle multiple virtual machines online. It was originally created and developed for the purpose of educational use in IT area. However, as the system grew, it began to be utilized in other projects and by other departments as well. (Nurmi, 2022.)

The project was started by Jaakko Nurmi in 2015 and it was originally started by building the Qemu-KVM hypervisor platform for the Cyber Game project. During development, many new ideas for improvement were generated and Virtual Lab was expanded to its own development project. (Nurmi, 2016.)

The variety of lab exercises can be created with Virtual Lab, ranging from simple home network to advanced data center architecture. All of the equipment and devices on the system are virtualized versions of those that exist in the real world, allowing users to perform practical exercises in the Virtual Lab. Since those virtualized devices have the same features as the physical ones, it is even possible to connect the virtual environment to a network of the real world, thereby expanding research and testing possibilities. (Nurmi, 2022.)

To access the Virtual Lab, all users need is a web browser with HTML capabilities and an Internet connection of at least 10 Mbit/s. This made it possible for students to access the lab exercise from anywhere, anytime. Students can now work on their own environment and devices, whereas previously labs were carried out with physical equipment and they had to share the devices with other students. (Nurmi, 2022.)

3.3 Operation of Virtual Laboratory

To start working in the lab environment, users have to create a new lab instance or open the already running one which was created previously. Instances are virtual work space allocated for the user with predefined amounts of resources, where one can create and run virtual machines. When creating a new instance, lifetime has to be selected, ranging from 4 hours to 180 days. When the lifetime expires, the instance will be deleted permanently.

Users can select a scenario from a variety of prepared templates such as data center networks, cybersecurity training, networking exercises, etc. They can also start an empty instance and build the scenario from scratch by adding the required devices by themselves.

There are many kinds of pre-configured devices in the system and users can add those virtual machines to the instance. Ready-made devices are available with default settings of suitable configuration for each. Creating or deleting virtual machines and changing device configuration is done through a management interface called Virtual Device Manager (Figure 2). Device settings such as the number of processor cores, storage volume or memory size can be changed afterwards if needed. It is also possible to upload their own disk images or virtual devices from the local folder.

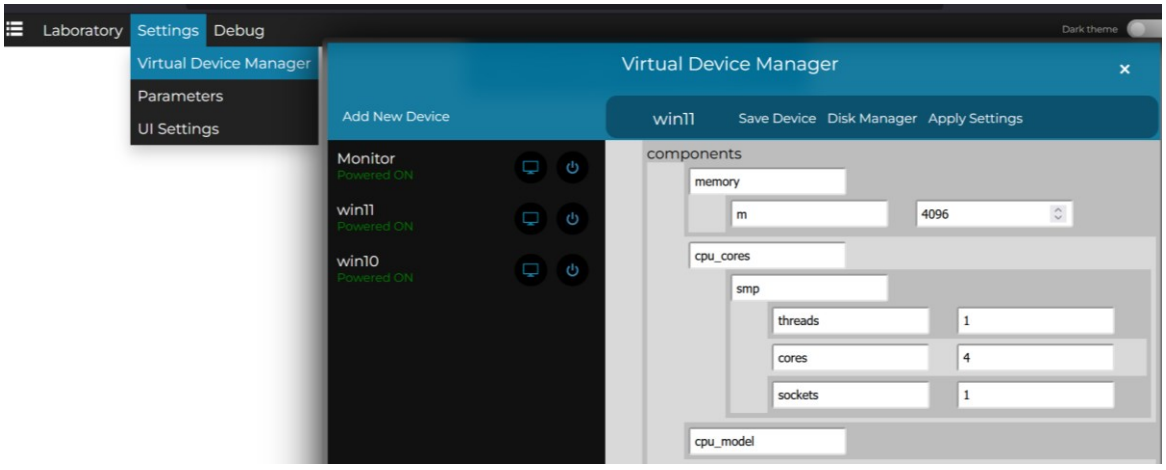

Figure 2. Virtual Device Manager

In order to operate and connect the virtual machines, those have to be added to graphical laboratory view, which is called network topology (Figure 3). In this view users can locate the devices as they want and connect them with virtual cables. Internet connection can be added as well.

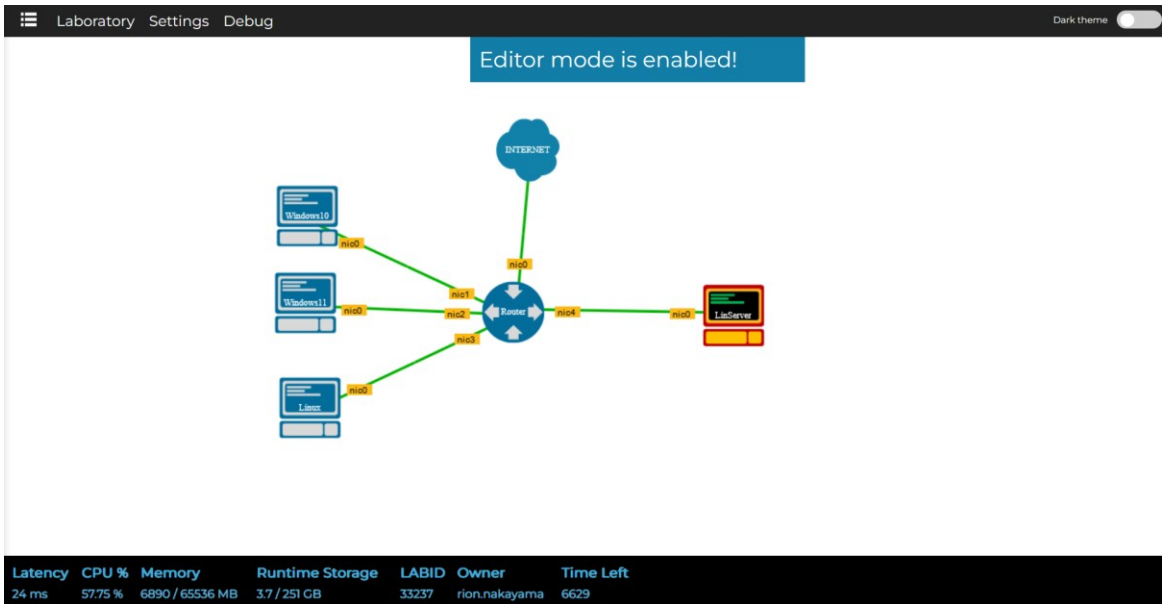

Figure 3. Network topology

Users can connect to the virtual machines by clicking the devices on the network topology. An interactive interface opens in a separate browser tab, allowing the users to work with the created virtual machine.

4 IMPLEMENTATION

Initially, it was planned to create the test environment with three node machines, which are Windows10, Windows11, and Linux. As browsers, Chrome, Firefox, Microsoft Edge were used. Only for Windows10, Internet Explorer was also installed. Later, Android was added as node.

VirtualBox was used to create the virtual machines, and the necessary software and browsers were installed on the virtual machines. It is also possible to create new virtual machines from an ISO file on Xamk Virtual Lab. However, it was chosen to install the operating systems and setup basic configuration using VirtualBox since it was easier for the first installation.

4.1 Set up Hub

Linux Xubuntu 22.04 LTS was used as a hub machine. Xubuntu is a light version of Ubuntu, which requires less memory and less-capable hardware. The reason for choosing Xubuntu was because it is open-source and simple to use. Moreover, since hub machine does not need much resources and the disk image had to be uploaded to Xamk Virtual Lab later, it was better to keep the virtual machine light weight.

In order to run the machine as a hub, Selenium server, java, and programing language to write Selenium scripts have to be installed. First, Open JDK was installed. For Selenium Grid, Java 11 or higher is required. Then Selenium server version 4.9.0 (the latest version at that time) was downloaded from the official web page of Selenium project. Python was chosen to write Selenium script since it was the most familiar language for the author. Therefore, Python program and Selenium library of it were also installed.

4.2 Set up Node

In order to run the machine as a node, browsers and browser drivers have to be installed, in addition to Selenium server and Java program to run it. Since it was aimed to test three operating systems: Windows10, Windows11 and Linux, these virtual machines were created with VirtualBox and required browsers were installed for each machine. The versions of each browser can be found in Figure 4. Depending on the browser versions, browser drivers from the official web pages were downloaded. For Chrome and Edge, there are specific drivers according to the browser version. For geckodriver for Firefox, the latest version available at the time and compatible with the browser version, 0.33.0, was installed. Then, path to the drivers to environment variables were added, so that the location does not have to be mentioned in the script. This makes it possible to manipulate the web drivers on several nodes without requiring that each machine has the drivers in the same location.

	Chrome	Firefox	Edge	Internet Explorer
Win10	114.0.5735.248	114.0.1	115.1901.183	IE mode in Edge
Win11	114.0.5735.248	114.0.1	115.1901.183	
Linux	112.0.5615.165	115.1.2	112.0.1722.64	

Figure 4. Browser versions

Once the hub and the node were ready, it was tried to see if the hub and the node could be connected. To run hub, following command is used: *java -jar selenium-server-<version>.jar hub*. The state of the server can be checked from the command line (Figure 5).

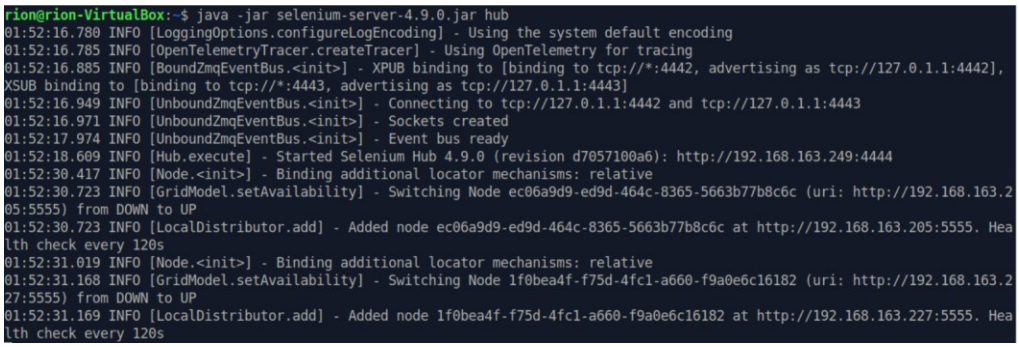

Figure 5. Start hub

For node, following command is used: *java -jar selenium-server-<version>.jar node --hub http://<hub-ip>:4444*.

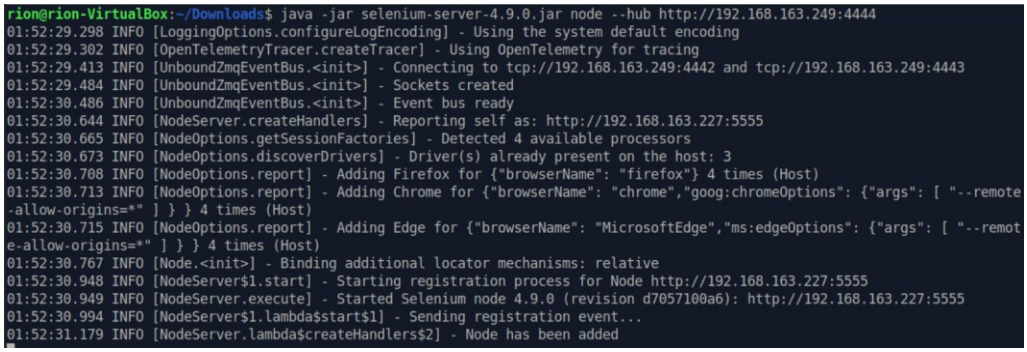

Figure 6. Start node

With command line, one can check that the node is added to the grid (Figure 6). In addition to that, Selenium Grid has web UI which can be accessed with

http://localhost:4444 on the hub, and one can check the information of nodes and session queue there.

4.3 Creating a scenario on Virtual Lab Environment

The first step in building the Selenium Grid environment in the Virtual Laboratory was uploading the VDI files to the personal inventory in the Virtual Lab. VDI (Virtual Disk Image) is a container format used by VirtualBox for hard disk of guest machines. In order to migrate virtual machines created with VirtualBox to Virtual Lab, VDI files can be used as hard disks of machines.

After VDI files are uploaded, virtual machines are created from those VDI files through the Virtual Device Manager. Since the feature of creating a new device from scratch does not work well yet, it was necessary to create a new device with a pre-installed virtual machine image, and swap the hard disk to the uploaded VDI file. Then, the number of cores on each machine was set to 4 and the memory to 4 GB.

OpenWrt is used as a router since it was pre-installed in the Virtual Laboratory, and connected the machines with the cables. Internet connection was also added to the scenario so that simple test scripts could be tested with some web pages. Figure 7 shows the network topology of the configuration.

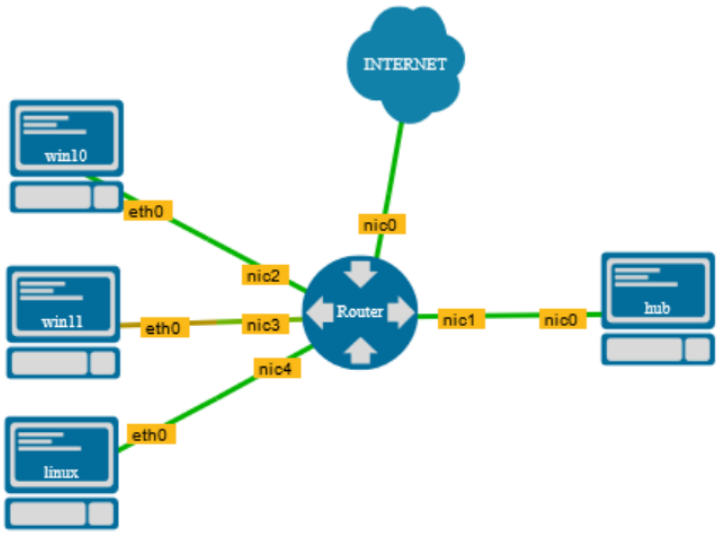

Figure 7. Hub-node setup in Virtual Laboratory

After adding all the required devices and connections, the hub machine and all node machines were started. Web UI shows three node machines and the browsers installed there (Figure 8).

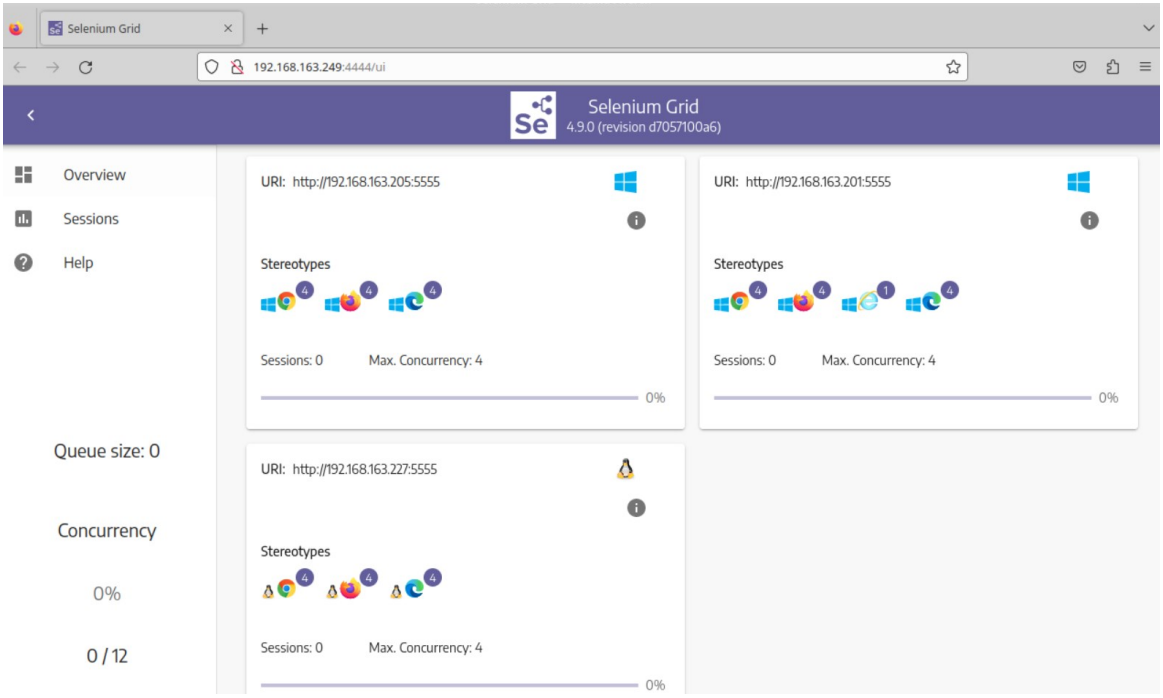

Figure 8. Three nodes connected to the hub

In the upper right of the browser icons, they have numbers of 4 and "Max Concurrency" is also displayed as 4, since each virtual machine has four cores. This means that the maximum number of simultaneous sessions of each browser is four. Number of Max Concurrency is the maximum number of browser automation sessions which can be started concurrently. For example, if Max Concurrency of chrome is four, four chrome windows can be automated at the same time. Default is the number of processors that are available.

4.4 Mobile testing

Once it was decided to use Appium for Android testing, it was attempted to setup the environment with Android Emulator of Android Studio and connect it with an Appium server, since it is the most popular and easiest way. Xubuntu virtual machine was created locally, and Android Studio and Appium server were installed there. Then, the VDI file of the virtual machine was uploaded to the

virtual environment and it was tried to run Android emulator to connect the emulator to Grid. However, it turned out that emulator works extremely slowly in the virtual environment. The performance could not be improved by simply increasing the amount of memory or number of cores. It is assumed that this is because the virtual environment itself is a nested virtualization environment, so running the emulator on a virtual machine involves too many layers and cannot be executed with sufficient quality.

Therefore, an alternate solution was tried, which is conducting testing with an Android x-86 virtual machine. Android-x86 is an open-source project aimed to port Android Open Source Project (AOSP) to the x86 platform. (Android-x86, n.d.) One can download the ISO file of different versions of android operating system and use it for creating virtual machines, or directly install it on the computer. In Virtual Laboratory there is a pre-installed android x-86 virtual machine which was created for the purpose of penetration testing lab exercises, so it was decided to use the preinstalled one. The version of Android was nine as it was the latest released version of the Android-x86 project. This is not the latest version of the android, but since the purpose here was to test the android and experiment with how the Selenium Grid would work in the virtual lab, this setup seemed sufficient.

In the setup, Android virtual machine, the computer which has Appium server and Selenium server installed, and the hub machine are required. The Android virtual machine is connected to the hub as a node, but a normal computer is also required in the setup as a role of bridge between Android and hub machine. This is because Android itself cannot have Selenium server and Appium server directly. For this function, an xubuntu virtual machine image which was preinstalled in the virtual lab with the name of “Mobile_Pentester” was used. The particular version of the preinstalled xubuntu machine was chosen because it was created for android penetration testing and already had some necessary tools installed. The network topology of the setup is shown in Figure 9.

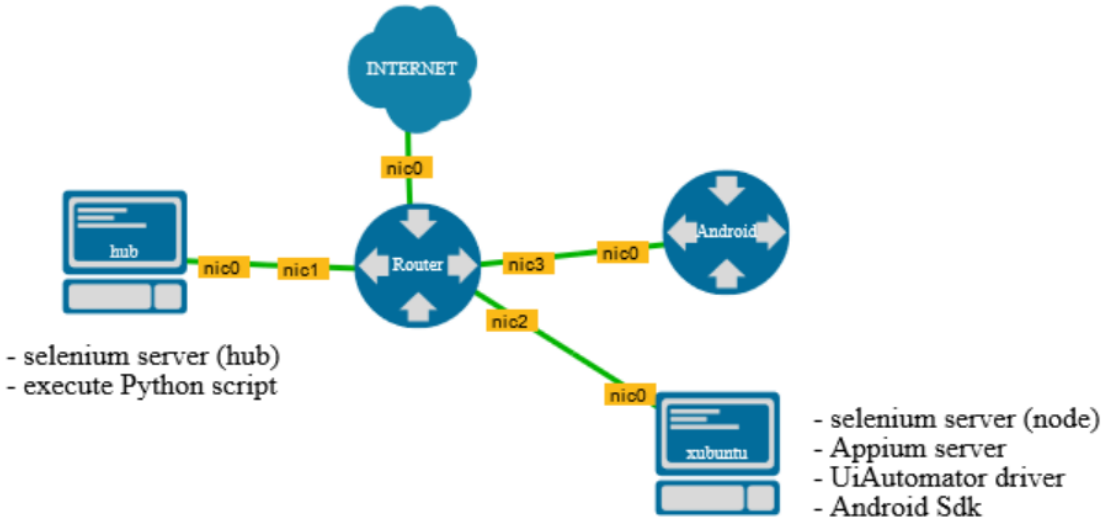

Figure 9. Mobile test setup

In order to setup the testing environment, Appium server first needed to be installed on the xubuntu virtual machine. The basic requirements for installing an Appium server are having Linux, macOS, or Windows operating system, Node.js version 14.17.0, 16.13.0, or 18.0.0 or higher is installed, and NPM version 8 or higher is installed. Appium server can be installed with a single NPM command; *npm i --location=global appium*. Next, UiAutomator2 Driver was installed, which is a specific Appium driver for Android devices. In order to use UiAutomator2, there are some requirements in addition to normal Appium requirement, which is setup of Android SDK Platform-Tools and Java JDK.

Android SDK Platform-Tools is one of the components of the Android SDK (Android software development Kit), which is developed by Google. It consists of software development tools and libraries that are used for developing android applications. It also contains build-tools, command-line tools, Android Emulator and so on. The main part of SDK Platform-Tools is ADB (Android Debug Bridge) and fastboot, which provides interface with the Android platform. It is included in Android Studio, but it is also possible to download only ADB command line tool if the user does not need Android Studio. (Android Studio, 2023a.)

ADB is a command line tool which can be used for communicating with devices running any version of Android to facilitate variety of actions such as installing or

debugging applications. One can even execute commands that are either impossible or hard to execute from the device itself. ADB works as a client-server program and is composed of three elements: client, daemon, and server. Client is the interface that runs on the development machine and sends commands to the devices or emulator, using command-line terminal. Daemon executes the commands received from the client on the device. The server also runs on the development machine and manages connection between the client and the daemon. (Android Studio, 2023b.)

For the SDK Platform-Tools to work properly, the environment variable ANDROID_SDK_ROOT or ANDROID_HOME must be set to indicate the path to the directory where the Android Tools are installed. In addition, Java JDK8 or higher is required, and the JDK home directory must be specified as the environment variable JAVA_HOME. Finally, the driver can be installed with a command: *appium driver install uiautomator2*.

To run Android as a Node machine, ADB client have to be started on the xubuntu machine first. Connection with an Android can be established with the following command*; adb connect <Android-ip>:5555*. Then, Appium server can be started with the command: *appium*. Before starting the Selenium server as a node, a configuration file has to be prepared (Figure 10). In Selenium 4, TOML file is used for describing configuration information.

```
[server]
port = 5556

[node]
detect-drivers = true

[relay]
url = "http://localhost:4723"
status-endpoint = "/status"
configs = [
    "1", "{\"platformName\": \"Android\", \"platformVersion\": \"9.0\", \"deviceName\
": \"Android\",\"browserName\":\"chrome\"}"
]
```

Figure 10. Configuration file of Android node

The configuration file specifies the port number that the selenium server listen on, URL of the Appium server, and the information of the device to which the call will be relayed.

After those preparations, selenium servers to run hub and node will be started. Hub can be started with the following command: *java -jar selenium-server-<version>.jar hub*. To start a node, IP address of the hub and name of the configuration file have to be specified: *java -jar selenium-server-<version>.jar node --hub http://<hub-ip>:4444 --config <filename>.toml*.

5 DEMONSTRATION

For the purpose of demonstrating the created environment, testing was performed against the MediaWiki web site (Figure 11). A MediaWiki server was created in the test scenario, and several tests were performed against the web pages.

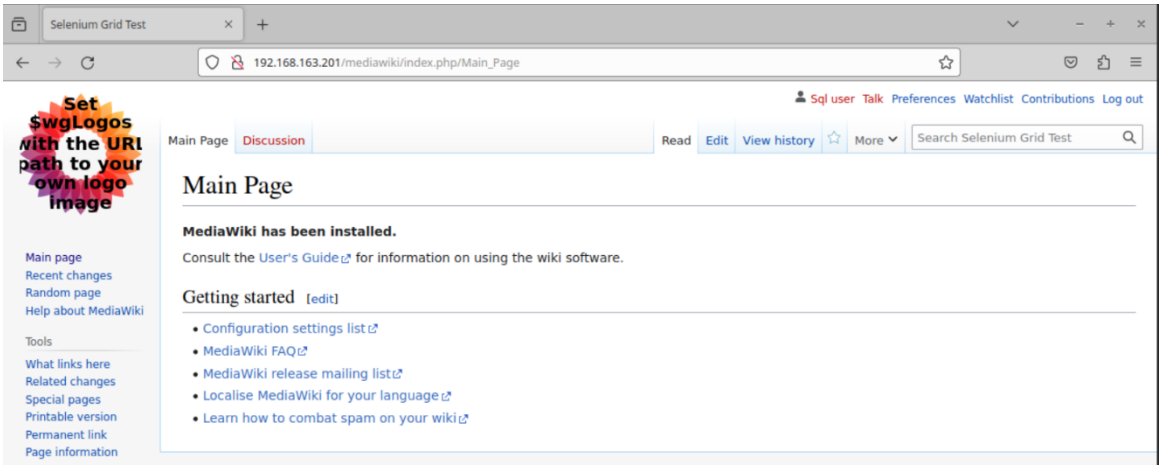

Figure 11. MediaWiki web page

MediaWiki is an open-source and free wiki software. It is server-based, scalable highly powerful and functionally rich software. Users can add extensions to customize it for ease of use. It uses PHP for showing and processing information stored in a database like MySQL. It is designed for large sites that are expected to receive a large amount of traffic and can withstand millions of hits per day. (MediaWiki, 2023.) MediaWiki was picked simply because it is free to use and

easy to setup. Additionally, adding new pages and features is also not complicated. The test target did not have to be a complex web page, but I needed a website with several actions and features so that I could try out different actions with automation.

Python was picked to write the scripts. Before running the program, the Selenium library needed to be installed with the following command: *pip install selenium*. The automated user actions were as follows:

1. Start the browser session and open chrome window.
2. Maximize the window.
3. Access to the media wiki page.
4. Verify that the page name is correct.
5. Type text “test1” into the search bar and press the search button.
6. Press button which navigate to create a new wiki page.
7. Input the description of the page and press save button.
8. Verify that the page title is correct and close the window.

5.1 Single session test

First, the test was run with only one node connected. The script shown in Figure 12 was written for automating the actions described above. The program gives errors if some assertion error occurs or it cannot find elements. However, in general, Selenium does not have the feature to return test results of whether it passed or failed. Therefore, it is better to include print sentences in case some detailed output is needed. In the script, print statements were used to indicate which action had been completed.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

hub = 'http://192.168.163.149:4444'
url = "http://192.168.163.201/mediawiki/index.php/Main_Page"

option = webdriver.ChromeOptions()

driver = webdriver.Remote(
    command_executor = hub,
    options = option
)

driver.implicitly_wait(5)
driver.maximize_window()
driver.get(url)
print("opened MediaWiki")

try:
    title = driver.title
    assert title == "Selenium Grid Test"

    driver.find_element(by=By.ID, value="searchInput").send_keys("test1")
    print("inserted text to search bar")

    driver.find_element(by=By.ID, value="searchButton").click()
    print("clicked search button")

    driver.find_element(by=By.CLASS_NAME,value="new").click()

    driver.find_element(by=By.ID, value="wpTextbox1").send_keys("This page is created for test")
    print("page information was entered")

    driver.find_element(by=By.ID, value="wpSave").click()
    print("clicked save button")

    message = driver.find_element(by=By.ID, value="firstHeading")
    value = message.text
    assert value == "Test1"
    print("successfully created page")

finally:
    driver.quit()
```

Figure 12. automation script of single test

The line “chrome_options = webdriver.ChromeOptions()" means that the session will open Chrome as the browser to automate. To be precise, the Chrome browser driver is selected here, so the Chrome driver is started and a Chrome session is initiated. The configuration of which browser to use can be easily modified by changing this line. For example, changing this line to "options = webdriver.FirefoxOptions()" will open the Firefox browser and do exactly the same action. This script does not describe the platform name, so it can be used on any operating system.

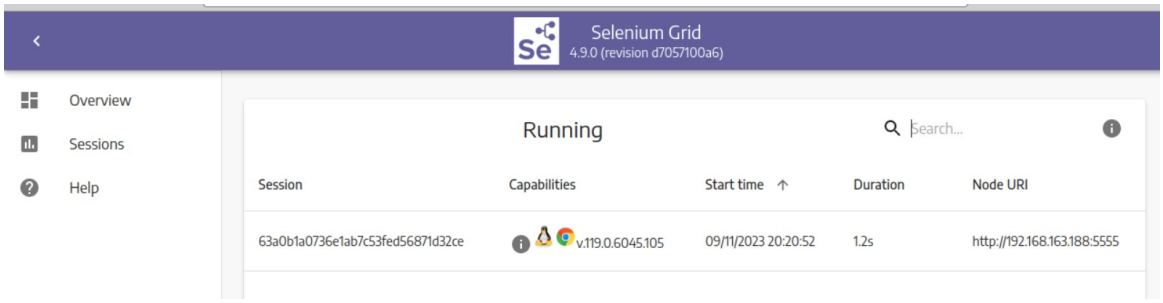

Figure 13. Session queue

Once the script is executed on hub, the session will be added to the new session queue. This session queue can be monitored on the GUI (Figure 13). The running session and other sessions in the queue can be checked. The session will be forwarded to the available node which meets the condition of the request. If there are several nodes that meet the requirement, one of them will be chosen randomly.

5.2 Parallel test

Next, it was tried to run the test in parallel, with several operating systems or with several different browsers. At first, the test was run with one specific browser and on three different operating systems. With this test, three nodes are connected to one hub.

It was needed to find out how to execute the test in parallel, since only connecting several node machines was not enough to run the test script on all of the three nodes. To execute test cases in parallel, it seemed that other frameworks or modules are used in most cases. For running Selenium with Java program, TestNG or Maven project, but Python is used in this study and these frameworks could not be used. Therefore, a testing framework for Python called "Pytest" was used.

Pytest is a framework for testing that supports creating test cases using Python. It helps to write readable and small tests easily, and also can be extended in order to provide support for complex functional testing of applications and libraries. The plugin of Pytest named "pytest-xdist" was used to execute the test script across

multiple node machines. The main purpose of pytest-xdist plugin is to distribute runs of tests across several processors for faster test execution. (pytest-xdist 2022.) It can be installed with the following command: *pip install pytest-xdist*. Pytest itself was not used when writing the test script, but the pytest-xdist plugin helped to distribute the test execution for three connected nodes. For example, "*pytest -2*" means that two processes will be started in parallel.

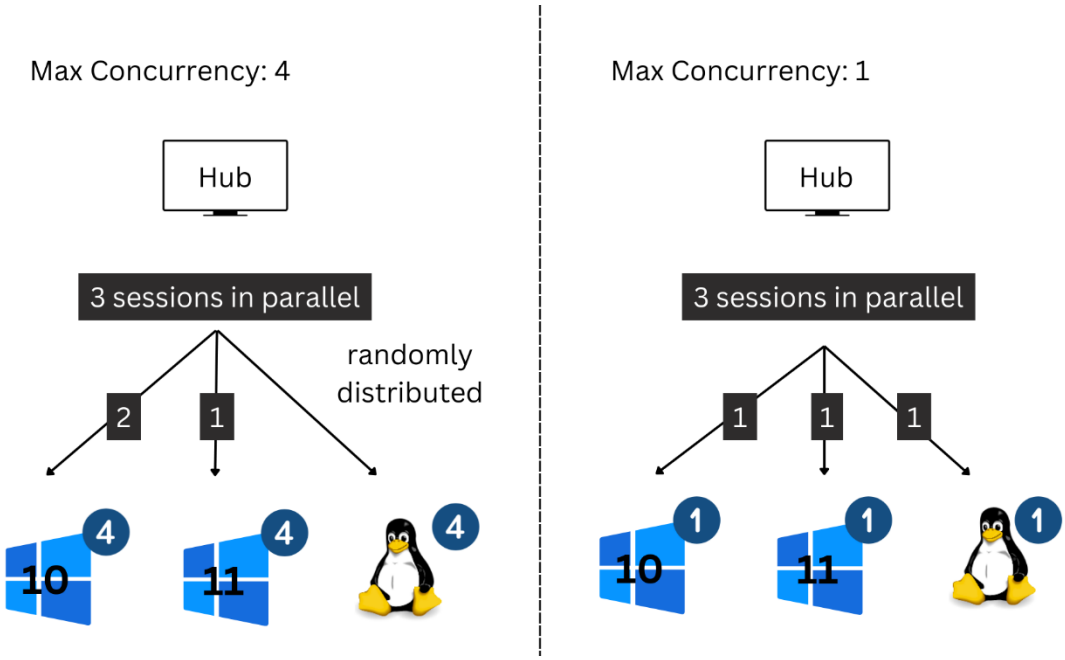

Figure 14. Parallel sessions distribution

However, simply increasing the number of sessions did not solve the problem in a straightforward manner. When a node is started without specifying the number of concurrent sessions, the maximum number of sessions for each node is set as the number of CPUs on each machine. With pytest-xdist, the processes will be distributed randomly across available CPUs. If three processes started when the max concurrency is four, those three sessions will be distributed to available nodes randomly. (Figure 14 left) It can start two sessions on Windows 10 and one session on Windows 11, or all three sessions might be forwarded to Linux. Therefore, it was necessary to limit the maximum number of sessions for each node. (Figure 14 right) Otherwise, the test execution would not be distributed to each node as desired, and multiple sessions could be started on a single node. Max concurrency can be set by specifying the command line option "*--max-sessions*" when starting a node.

```python
from selenium import webdriver
from selenium.webdriver.common.by import By

hub = 'http://192.168.163.148:4444'
url = "http://192.168.163.201/mediawiki/index.php/Main_Page"

option = webdriver.ChromeOptions()

driver = webdriver.Remote(
    command_executor = hub,
    options = option
)

name = driver.capabilities['platformName']

driver.implicitly_wait(5)
driver.maximize_window()
driver.get(url)

if name == 'LINUX':
    page = "test linux"
    info = "This page is created on linux"

elif name == 'Windows 10':
    page = "test win10"
    info = "This page is created on windows10"

elif name == 'Windows 11':
    page = "test win11"
    info = "This page is created by windows11"

try:
    title = driver.title
    assert title == "Selenium Grid Test"

    driver.find_element(by=By.ID, value="searchInput").send_keys(page)
    driver.find_element(by=By.ID, value="searchButton").click()
    driver.find_element(by=By.CLASS_NAME,value="new").click()
    driver.find_element(by=By.ID, value="wpTextbox1").send_keys(info)
    driver.find_element(by=By.ID, value="wpSave").click()

    message = driver.find_element(by=By.ID, value="firstHeading")
    value = message.text
    assert value == page.capitalize()

finally:
    driver.quit()
```

Figure 15. Script for parallel execution

The script which is used for the parallel testing can be seen in Figure 15. The action the code is automating is same as the first test with single session. However, since the same script would be executed on different three nodes, I had to make some changes on the script. The process of creating a new page on MW is automated, but it is not possible to create a page with the same name in three different sessions. Each session must create a different page.

Once the session is initialized on each node, it detects on which operating system the program is running. The line “name = driver.capabilities['platformName']” performs this action. Then, there is conditional branching depends on the platform name it has detected. For instance, the page named “test linux” will be created with the session on the Linux machine.

As the whole process to execute the parallel test on three nodes connected, there were following steps:

1. Start the hub.
2. start the nodes with the command: *java -jar selenium-server-<version>.jar node --hub http://<hub-ip>:4444 --max-sessions 1*
3. execute the test script on the hub machine with the following command: *python3 -m pytest -n 3 <filename.py>*

Furthermore, I tried to run three different operating systems and three different browsers (Chrome, Firefox and Microsoft Edge) at the same time. For this execution I used the same script which was used for three sessions in parallel (Figure 15). The script is for automating Chrome browser. So, I just changed the line of browser option to Firefox and Microsoft Edge, and made three separate python script for each browser. Then, created shell script to execute those three scripts together (Figure 16).

```
python3 -m pytest -n 3 firefox_newpage.py &
python3 -m pytest -n 3 ms_newpage.py &
python3 -m pytest -n 3 chrome_newpage.py
```

Figure 16. Shell script for nine sessions simultaneously

As three sessions is occurring on one node machine at the same time, each node have to have at least three CPU cores for running this test. Although there were small time difference depends on the browser, it was smooth to run the tests.

5.3 Waiting strategy

During the experiment there were many cases that the test run failed because the next Selenium command is executed before the web page is fully ready to execute the next command.

Selenium has its own "ready state" value which is used for detecting if the web page has finished loading or not, and usually the navigation commands wait until the page is ready. However, the "ready state" is based on only HTML loading assets and does not concern JavaScript loading assets, while it can make changes to the application. This sometimes causes a situation where the web elements that have to be found by the driver are not yet ready when the Selenium command is executed. In many web pages, elements are added dynamically by some action or appearance will be changed by clicks. In order for Selenium to communicate with the element, the element has to be present and displayed on the web page. (Software Freedom Conservancy, n.d.b.)

In the beginning, I was using sleep command to solve this problem, but it is difficult to estimate the enough amount of waiting period. On the contrary, if the time is too long and added after every action, it makes the duration of the session too long.

Implicit wait method was the perfect method to solve this problem. It is a built-in tool of Selenium for automatically waiting for elements to be ready. With Python it can be assigned by the following command: *driver.implicitly_wait(2)*. With this example it will wait for maximum 2 seconds until elements appear. Once the command is invoked in the script, it applies to all of the element finding calls for the one entire session. The driver will continue to the next code execution as soon as the element is located.

5.4 Mobile testing

I tried to automate the same action described in page 26, which is creating new page on MediaWiki. In order to execute the test on Android, almost the same

script can be used for normal sessions, but some specific configuration was required in the script (Figure 17).

```
caps = {
        "platformName":"Android",
        "appium:automationName":"uiautomator2",
        "deviceName":"Android",
        "browserName":"chrome"
        }

hub = 'http://192.168.163.213:4444'

driver = webdriver.Remote(hub,caps)
```

Figure 17. Mobile test script

Chrome browser was automated here. In addition to the script for normal node, information of the Android device should be addressed to start the driver. Also, in order to automate chrome browser, chrome driver for appium is required. It can be installed by running the following command on the computer which has appium server: *appium server --allow-insecure chromedriver_autodownload*.

6 COMPANY TESTCASE

During the QA project, we cooperated with a Finnish IT company and conducted UI test against the company's web application, using the setup of Selenium Grid and Xamk Virtual Environment. The company allowed us to test the beta version of their online education platform. The web application was based on Moodle, which is a learning management system used for creating and maintaining online courses. It was decided to do an automation test of some basic features of the application. The basic daily interactions that users would have with the application was automated, such as creating courses, registering new users, enrolling users to specific courses, submitting material and so on.

The test was mainly conducted across three operating systems: Linux, Windows 10 and Windows 11. The browsers that were used were Chrome, Firefox, and Microsoft Edge. The company provided us a URL to access the web application, which meant that the web page had to be accessed via the Internet, unlike the demonstration with MediaWiki. Compared to the tests using MediaWiki, it was

found that these tests tended to fail a bit more frequently, even though the web page was working properly. Especially, the greater the number of tests executed simultaneously in a parallel test, the greater the percentage of failures. The main reason for the failure was that the driver could not find the web element due to the excessive time it took to load the web page. It is assumed that this is because of the fact that the site had to be accessed via the Internet, and that the more complex actions were automated using longer test scripts compared to the MediaWiki tests. To make this better, it was tried to improve the script by including explicit wait and sleep commands, or simply by testing it in a good Internet environment.

7 DISCUSSION

Throughout the whole process, it appeared that the combination of Selenium Grid and Virtual Laboratory worked quite successfully, although there were some difficulties in working on Virtual Lab.

The most significant advantage of using Virtual Lab environment for Selenium Grid testing would be the possibility and the easiness of handling multiple virtual machines. As the biggest benefits of Selenium Grid is the simplicity of testing various configurations and the possibility of parallel testing, necessity of several or many node machines is inevitable. While there is a limit of numbers of virtual machines that can be run locally, one can create and handle lots of machines easily. Utilizing the Virtual Laboratory enables users to create and network multiple virtual machines smoothly, which makes it highly adaptable and capable of working with Selenium Grid. Also, a visual network topology-based GUI simplified the process. Intuitive operation made it smooth to build environments with multiple virtual machines and to run several virtual machines simultaneously. Since additional virtual machines can be quickly incorporated into the scenario, the scope of testing can be easily extended, such as by creating nodes that use different versions of browsers or integrating other platforms.

Various kinds of preinstalled machines made the process smooth as well. Since Virtual Laboratory is being used for lots of different kinds of lab exercises and

studies, it has a wide range of virtual machines with different setup and scenarios. These ready-made machines or environments can be helpful for building variety different test environments, depends on the test scope. Preinstalled android and Linux machine created for penetration testing was used for building Mobile testing environment, which reduced the workload greatly.

Furthermore, the feature of saving crated virtual machine made the working process smoother. The feature can be used as a snapshot to save the virtual machine on the specific state, or duplicate the machine. With Virtual Laboratory saving image of created virtual machine and copying them were so simple.

On the other hand, there were some obstacles as well. Since the Virtual Laboratory is still under development, some features were not yet ready, or were a bit laggy or slow sometimes. One problem that happened several times during working on the Virtual Laboratory was that other hard drivers could not be added for getting additional storage of virtual machines. There were some cases that storage was running out but could not be added later, so the machine had to be rebuilt from scratch. To increase the storage, the only way is recreating the device and start the machine from VDI file or boot up from ISO file, which means losing the work on that machine.

When testing was conducted with MediaWiki server, there was not much concern about network speed that much since the web server was also located in the virtual laboratory. However, while testing the company's web application, the network speed seemed to affect the tests. If the web application under test had to be accessed via the Internet, there might be times that the web page takes a long time to load and the test does not go well. External factors such as internet connection quality and the script's implementation can influence test results. Therefore, manual testing would be required along with automated testing, especially when automated tests encounter failures.

8 CONCLUSION

The purpose of this thesis was to examine how Selenium Grid and Virtual Laboratory work together, and what kind of effect it has on testing. To prepare the testing environment for Selenium Grid, multiple machines are required to setup hub - node relationships. It was hoped that Xamk Virtual Lab environment would make the testing process efficient and easy, due to its ability to build and handle a variety of virtual machines on the cloud.

The scenarios for Selenium Grid testing of three computer operating systems (Windows 10, Windows 11 and Linux) and Android were created in the virtual environment. Appium was used to add the Android virtual machine as a node to the Grid. Test scripts were prepared to automate user actions on the MediaWiki web page and several tests were performed, including parallel execution. As a part of the Quality Assurance workshop, automation testing was conducted on the beta version of company's web application as well.

Virtual Lab environment enabled to build the Selenium Grid testing environment with several node machines, and the intuitive operation with network topology allowed smooth handling of many virtual machines. While there are some challenges to working with the Virtual Laboratory, it can be said that the Virtual Lab overcomes the drawbacks of the Selenium Grid quite well.

On the other hand, it was found that test runs became unstable and tended to produce inconsistent results, with test cases with the company application. This was presumably because the website had to be accessed via the Internet and more complex behaviors were automated. The same can be said for all automated testing, but it is essential to combine it with manual testing, especially in the execution of larger and more complex test cases. With the current implementation, simple tests such as short scripts or single sessions are relatively reliable. For large test cases where many scripts are executed in parallel, the environment is not yet satisfactory. Therefore, stability has to be improved by analyzing the causes of test instabilities.

Various elements affect the quality of test execution, such as internet connection, test script, configuration of each machine, and test execution method. The testing environment can be improved by monitoring test execution and analyzing failure patterns to reduce inconsistencies in test results. Since the foundation for the testing has been constructed through the study, the environment can be developed to accommodate a larger testing scope by increasing the number of nodes to expand the test coverage or by improving stability.

REFERENCES

Android Studio. 2023a. Command-line tools. Web page. Available at: https://developer.android.com/tools [Accessed 15 November 2023].

Android Studio. 2023b. Android Debug Bridge (adb). Web page. Available at: https://developer.android.com/tools/adb [Accessed 15 November 2023].

Android-x86. n.d. Android-x86 Run Android on your PC. Web page. Available at: https://www.android-x86.org/ [Accessed 15 November 2023].

Appium Documentation. n.d. Appium Project History. Web page. Available at: http://appium.io/docs/en/2.1/intro/history/ [Accessed 23 September 2023].

AWS. n.d. What's the Difference Between Type 1 and Type 2 Hypervisors?. Web page. Available at: https://aws.amazon.com/compare/the-difference-between-type-1-and-type-2-hypervisors/ [Accessed 24 September 2023].

Chaubal, P. 2018. Selenium WebDriver Quick Start Guide. Birmingham: Packt Publishing Ltd. Ebook. Available at: https://ebookcentral.proquest.com/lib/xamk-ebooks/detail.action?docID=5594251 [Accessed 24 August 2023].

Gundecha, U. & Avasarala, S. 2018. Selenium WebDriver 3 practical. Birmingham: Packt Publishing Ltd. Ebook. Available at: https://ebookcentral.proquest.com/lib/xamk-ebooks/detail.action?docID=5485029 [Accessed 24 August 2023].

Holger krekel and contributors. 2022. pytest-xdist. Web page. Available at: https://pytest-xdist.readthedocs.io/en/stable/ [Accessed 16 November 2023].

IBM. n.d. What is virtualization?. Web page. Available at: https://www.ibm.com/topics/virtualization [Accessed 23 September 2023].

Jantunen, S. 2023. Ohjelmistoalan opetuksen yritysyhteistyötä kehittämässä. South-Eastern Finland University of Applied Sciences. PDF document. Available at: https://urn.fi/URN:ISBN:978-952-344-527-7 [Accessed 18 November 2023].

Media Wiki. 2023. Manual:What is MediaWiki?. Web page. Available at: https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F [Accessed 15 November 2023].

Nurmi, J. 2016. Implementation of Nested Virtual Laboratory System. Kymenlaakson ammattikorkeakoulu. Information Technology. Bachelor thesis. PDF document. Available at: https://www.theseus.fi/handle/10024/107061 [Accessed 23 September 2023].

Nurmi, J. 2022. VirtualLab – more than a traditional simulator. Web page. Available at: https://read.xamk.fi/2022/digitaalinen-talous/virtuallab-more-than-a-traditional-simulator/ [Accessed 24 September 2023].

Oracle. n.d. Oracle VM VirtualBox. Web page. Available at: https://www.oracle.com/virtualization/virtualbox/ [Accessed 24 September 2023].

Portnoy, M. 2016. Virtualization Essentials. Indiana: John Wiley & Sons, Incorporated. E-book. Available at: https://ebookcentral.proquest.com/lib/xamk-ebooks/reader.action?docID=4644086 [Accessed 23 September 2023].

Software Freedom Conservancy. n.d.a. Selenium History. Web page. Available at: https://www.selenium.dev/history/ [Accessed 24 August 2023].

Software Freedom Conservancy. n.d.b. Documentation. Web page. Available at: https://www.selenium.dev/documentation/ [Accessed 28 August 2023].

Stewart, S. & Burns, D. 2018. WebDriver. Web page. Available at: https://www.w3.org/TR/webdriver1/#references [Accessed 28 August 2023].

vmware. n.d. What is a hypervisor?. Web page. Available at: https://www.vmware.com/nordics/topics/glossary/content/hypervisor.html [Accessed 23 September 2023].

LIST OF FIGURES