Creating Pipeline and Automated Testing on GitLab

Saad Turky

Master's Thesis
March 2021
Information and communication technology
Master's Degree Programme in Full Stack Software Development

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s) Turky Jgeif, Saad	Type of publication Master's thesis	Date March 2021
		Language of publication: English
	Number of pages 93	Permission for web publication: x
Title of publication **Creating Pipeline and Automated Testing on GitLab**		
Degree programme **Full Stack Software Development**		
Supervisor(s) **Salmikangas, Esa & Huotari, Jouni**		
Assigned by Nestronite Oy		

Abstract

It was identified by the assigner Nestronite Oy that the main objective was to create a modern software development environment for the cloud-based application called Jaxber. That includes continuous integration, deployment (CI/CD), and automated testing. The value of software testing is significant. Nevertheless, software development environments can create time and cost limitations that make it hard to completely test an application preceding release. If faults slip unnoticed into the production environment, the end result can be customer dissatisfaction and increased maintenance costs.

The goal of the author was to create the required environment. The author studied the requirements of creating a pipeline, Docker, Robot Framework, and its libraries. The focus was on how to test the application using the Robot Framework on GitLab. The author used a diary-based method where he explained his work by writing everything about his learning during the period of writing. The relevant data for the qualitative research was gathered on a daily basis and followed up by a weekly reflective summary. The reporting time covered nine weeks in calendar time.

Based on the results, the automated test, building, and deployment were achieved successfully. Where the development environment created a pipeline containing three stages. The first stage was Docker. In this stage, the author used Dockerfile for production. In the second stage, the author deployed the application on GitLab pages in order to test it using Robot Framework. The last stage was the test using Robot Framework, where the author created a test cases file at the root of the application, then called it in this stage.

Related to GitLab, the author created a virtual machine. On this virtual machine, the author set up the GitLab runner to run the pipeline, in addition to all the packages that could be installed on this virtual machine to reduce the time of the test.

Keywords/tags (subjects)
Containers, Docker Engine, Docker architecture, Robot Framework, GitLab, Continuous integration

Miscellaneous (Confidential information)

Contents

Figures

1. Introduction

1.1 Background and motivation

Nestronite Oy, which was founded in 2011, participated in the development of FreeNest. FreeNest is an open source-based platform aimed to streamline teamwork and project management. In order to improve their operational capabilities, Nestronite Oy needs to create a modern software development environment for the cloud-based application called Jaxber. That includes continuous integration, deployment (CI/CD), and automated testing.

The value of software testing is significant. Nevertheless, software development environments can create time and cost limitations that make it hard to completely test an application preceding release. If faults slip unnoticed into the production environment, the end result can be customer dissatisfaction and increased maintenance costs.

Test automation permits your team to implement more tests in less time, improving coverage and delivering human testers to do more advanced, examining testing. Automation is especially useful for test cases that are implemented continually.

The company has a development environment on GitHub, and it wants to expand to test its applications in different environments like GitLab.

CI and CD are two abbreviations often used in modern development practices and DevOps. CI refers to continuous integration, an essential DevOps best procedure where developers regularly combine code changes into a central repository where automated builds and tests run. But CD can either mean continuous delivery or continuous deployment.

Continuous integration sets great importance on testing automation to make sure that the application is not broken-down whenever new commits are integrated into the main branch.

Continuous delivery is continuous integration's extension because it deploys all code changes automatically to the testing and/or production environment after the stage of the build.

The motivation for the choice of subject is to learn new and modern technologies. Learning to build a pipeline includes many factors, the most important of which are

Docker, Robot Framework, and git. Learning these factors will increase the skills of the author, especially the skills that are required to find a job these days.

1.2 Research Focus

The goal of the author is to create a modern software development environment for the cloud-based application called Jaxber. That includes continuous integration, deployment (CI/CD), and automated testing. The main reason to select this research is to learn modern technologies such as Docker and Robot Framework framework, which are among the most important modern technologies to test applications besides that will help the author to develop his skills.

1.3 Research Method

In this research, the author utilizes a diary-based method where he writes down his learning and problem-solving. The relevant data for the research is gathered during June–July 2020 on a daily basis and followed up by a weekly reflective summary. The reporting period covers approximately 60 days in calendar time. It should also be noted that chapters two and three are written using the first person due to the diary method.

Daily Diary Methodology is a group of valuation methods that let researchers to review individuals' experiences, behavior, and circumstances in natural settings, in or on the brink of real-time, and on repeated measurement occasions over an outlined period (ranging from a couple of days to months). Semantically, the term "daily diary methodology" implies a once-per-day evaluation method. Often, the term is employed in a wider sense to incorporate methods assessing individuals at multiple times per day (also called "experience sampling"; Hektner, Schmidt, & Csikszentmihalyi, 2007).

Additionally, the term "daily diary methodology" indicates the utilization of some kind of diary (paper-and-pencil or electronic) and is thus constrained to self-report data. Recently, however, the scope of intensive repeated measurements in naturalistic settings has broadened to incorporate also non-self-reported aspects of everyday experience, like psychophysiological status, physical activity, and auditory environment.

“A fundamental benefit of diary methods is that they permit the examination of reported events and experiences in their natural, spontaneous context, providing information complementary to that obtainable by more traditional designs (Reis 1994).”

Another is the dramatic reduction in the probability of retrospection, accomplished by reducing the amount of time elapsed between an experience and the account of this experience. Diaries offer the field of psychology with a powerful group of methods for studying different human phenomena involving personality processes (e.g., Bolger & Zuckerman 1995, Fabes & Eisenberg 1997, Rhodewalt et al. 1998).

Daily diary methods are the best when the research question aims at capturing representations of momentary experience or episodic memory of experiences. When the research question aims at capturing generalized beliefs (semantic memory), global or retrospective self-reports should be selected (Conner, Barrett, Bliss-Moreau, Lebo, & Kaschub, 2003).

2. Diary Reporting

2.1 Week 1

2.1.1 Wednesday 3 June 2020: Introduction

Today I will start writing my thesis which is related to pipeline and test automation. One of the most important requirements of the pipeline is called the Docker file. So, for today I planned to read about what is Docker especially? And containers in general, because I had no idea about containers before.

I read about Docker and found that very deep and interesting. There are thousands of articles, research, and courses about it.

The developer usually works on different projects in several languages, and each project requires different packages or system variables (Environment Variables) from the other project. Sometimes you use specific libraries or programs within your project that depend on various versions of the same package, in addition to the possibility that the developer may face difficulty in managing the packages, programs and services that work in his device because of the abundance or multiplicity of copies that he needs. He may also face some problems when transferring the project from the device used for development, to the device on which the program will work, or the server that will run this program. The program may not work in the production environment (Production) due to the difference in packages or system variables between the production environment and the development environment. In such a case, the developer needs more time to discover the problem and determine the package that he needs or the variable that needs to be prepared in the production environment.

Because of these and other problems, some developers resort to running projects on virtual machines in which they simulate the production environment, and this method also has some problems. Sometimes it may be difficult to simulate the production environment, or it may take a long time to prepare the environment on the virtual machine, in addition to the inability to isolate incompatible packages with each other within the virtual environment easily. This problem cannot be solved by using the virtual environment and running it on the server and using it in production because the virtual devices consume a lot of device resources, and the virtual device needing a long time during operation. All of the problems above can be solved by using Docker.

2.1.2 Containers

Containers are an implementable unit of software. In it we can package the application code, besides its libraries and dependencies, which help us to run it anywhere, desktop, traditional IT, or the cloud.

To do this, containers benefit from a form of operating system virtualization in which they take advantage of the features of the operating system to isolate processes and control the size of CPU, memory, and the disk that those processes have access to.

Containers are small, work quickly, and portable because, different from a virtual machine, containers do not need OS in each instance and can, instead, simply to get maximum advantage of the features and resources of the host OS.

Containers appeared for the first-time decades ago with versions like FreeBSD Jails and AIX Workload Partitions, but most modern developers consider 2013 as the start of the modern container era with Docker. (Containers, July 2020).

Finally, I can say I understand what containers are or what Docker is, was not difficult but required working hard and spending a lot of time to read about this important matter. I can say that I achieved my goals for today.

2.1.3 Thursday 4 June 2020: Benefits of containers

Yesterday went very well, where I understood the general concept of containers and Docker. Today I felt it is important to learn more about containers, so I decided to learn why we need containers or what are the benefits we can gain from containers. In addition, we should learn the cases of using the containers. I can briefly explain the most important benefits of containers as follows:

Light weight: Containers use the kernel of machine OS, canceling the need for a complete OS instance per application and making container files small and easy on resources. The smaller size of containers, compared to virtual machines, make them spin up rapidly and support for native cloud applications that scale horizontally better.

Portable and independent platform: Containers hold all their dependencies with them, which means that we can write the software one time and then run it without having to reconfigure it via laptops, cloud, and internal computing environments.
Supports modern development: Because of a mixture of its deployment portability/ consistency via platforms and its small size, containers are fit to patterns of modern development and applications, for instance: DevOps, serverless, and microservices, which are built by using code deployments in small increments.
Improved use: Like VMs before them, containers allow developers and operators to improve CPU and memory use of physical machines. Where containers can do further like, enable microservice architectures, the components of the application can be deployed and scaled precisely. (Containers, July 2020).

2.1.4 The cases of using containers

Containers became increasingly important, especially in cloud environments. Many companies are thinking of using containers as a replacement of VMs as a platform for their applications and workloads. But within this scope, there are key utilization cases of containers.
Microservices: Containers are small and lightweight, which makes them a decent counterpart for microservice structures where applications are developed of many loosely coupled and freely deployable littler administrations.
DevOps: the mixture of microservices as architecture and containers as a platform may be a common foundation for several teams that embrace DevOps because of the way they build, ship, and run the software.
Hybrid, multi-cloud: Because containers can run consistently anywhere, via laptop, on-premises, and cloud environments, they're a perfect underlying architecture for hybrid cloud and multi-cloud scenarios where organizations find themselves operating across a mixture of multiple public clouds together with their own data center.
Application modernizing and migration: one among the foremost common approaches to modernizing applications starts by containerizing them to migrate to the cloud. (Containers, July 2020).

I think today I learned important things about containers and the cases of their using because if you need to understand any new subject, you should read and go deeper in order to gain a comprehensive understanding of the topic.

2.1.5 Friday 5 June 2020: Docker Engine

During my reading yesterday, I noticed that there was an important thing to start understanding Docker; it was Docker engine. I specified today for the Docker engine only. I think it won't be a long day.

Docker Engine is a client-server application with the following significant segments:

- A server is a sort of long-running project called a daemon procedure (the Docker command).
- A REST API that determines interfaces that projects can use to converse. With the daemon and teach it what to do.
- A command-line interface (CLI) client (the Docker command).

Below are the Docker engine components, as shown in figure 1.

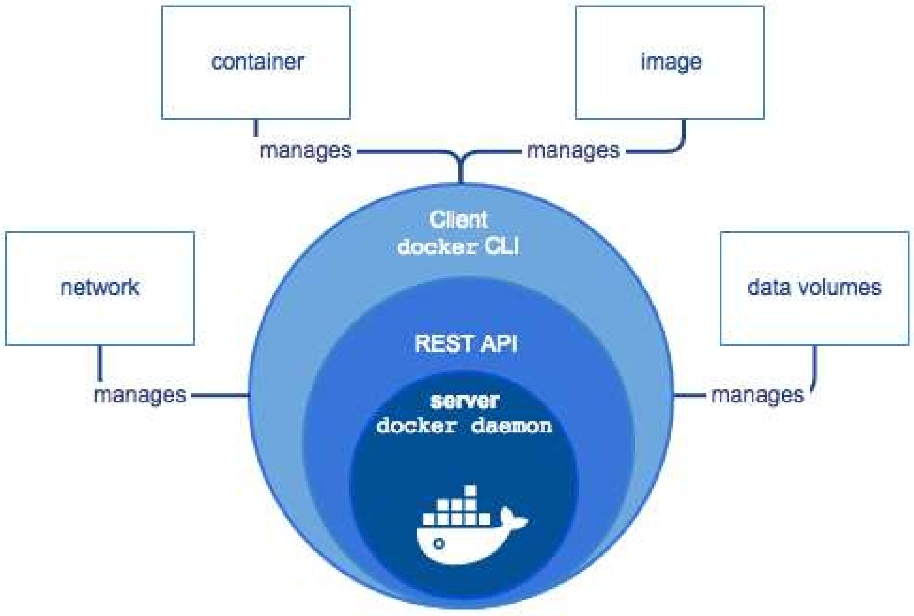


Figure 1. Engine components flow.

Source: https://docs.docker.com/get-started/overview/

The CLI utilizes the Docker REST API to control or connect with the Docker daemon through direct CLI orders. Numerous other Docker applications utilize the hidden API and CLI. The daemon makes and oversees Docker objects, for example, pictures, holders, systems, and volumes.

After reading in detail about the Docker Engine, I can say that today's goal was achieved. I also found that it is important to read about Docker architecture and the uses of Docker, so I will address these two topics tomorrow.

2.1.6 Saturday 6 June 2020: Docker architecture

My objectives for today are:

- Docker architecture
- The uses of Docker.

Both are related to understanding more about Docker. I think I need to spend a lot of time to read and understand the Docker architecture very well.

Docker utilizes client-server engineering. The Docker client converses with the Docker daemon, which does the hard work of building, running, and conveying your Docker containers. The Docker client and daemon can run on a similar framework, or you can interface a Docker client to a remote Docker daemon. The Docker client and daemon convey utilizing a REST API over UNIX attachments, or a system interface. In figure 2 presents the Docker architecture diagram.

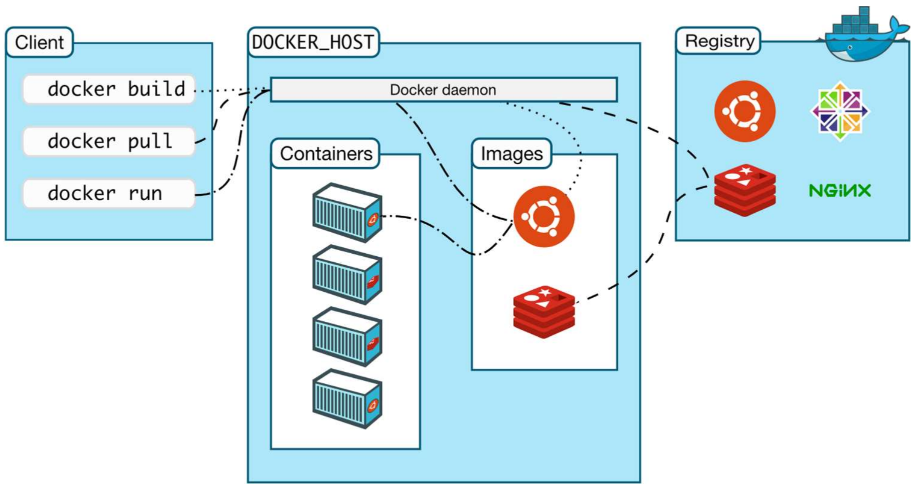


Figure 2. Docker architecture.

Source: https://docs.docker.com/get-started/overview/

2.1.7 The uses of Docker

Fast and consistent application delivery: Docker simplifies the development lifecycle by permitting developers to work in unified environments using local containers that give your applications and services. Containers are important for continuous integration and continuous delivery workflows.

Consider the following example:

- Developers write code and share their work with colleagues using Docker containers.
- They use Docker to push their applications into a test environment and implement automated and manual tests.
- When developers find errors, they can fix them in the development environment and redeploy them in the test environment for testing and verification.
- When the test is completed, getting the fix to the customer is as simple as pushing the updated image into the production environment.

Responsive deployment and scaling: Docker’s container-based platform allows for portable workloads. Docker containers can work on a developer’s laptop, on virtual machines during a data center, on cloud providers, or during a mixture of environments.

Docker’s portability and lightweight nature also make it easy to dynamically manage workloads, scaling up applications and services as business needs dictate, in near real-time.

Running more workloads on the same hardware: Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high-density environments and for small and medium deployments where you need to do more with fewer resources.

I got a lot of information about today's goals. Everything was new for me; in addition to I faced some difficulties with Docker architecture. After the reading, I can say that I have the base to go through deeper with Docker.

2.1.8 Sunday 7 June 2020: The concept of Docker

The objectives that I set for today are the concept of Docker to define Docker specifically and to know the difference between container and virtual machine. I think they are important and essential to learn more about Docker.

Docker is a tool intended for system managers and developers, used in building and running programs within a specific environment by running the project inside Docker Containers. Which facilitates the process of moving the project from one device to another. Without the need to modify system settings and variables or install or download specific packages, where only Docker is installed, and pre-made containers are run with specific settings that allow the application to run within these containers. A "Docker image" is a stack or collection of layers that are created from sequential instructions on a Dockerfile. The layers are "Read-Only." Although that is an exception of the topmost layer, which is "Read-Write" type. The Docker images can be recognized either by their "unique image ID," which is provided by Docker, or a convenient name or tag, which is provided by us means users. Finally, users can push or pull them from Docker Hub.

2.1.9 A Container and the difference between it and the virtual machine

A container is a group of packages isolated from your local device. You can run and use the programs inside the container without the packages inside the container interfering with the packages already installed on your device. The components of the Docker container are presented in figure 3.

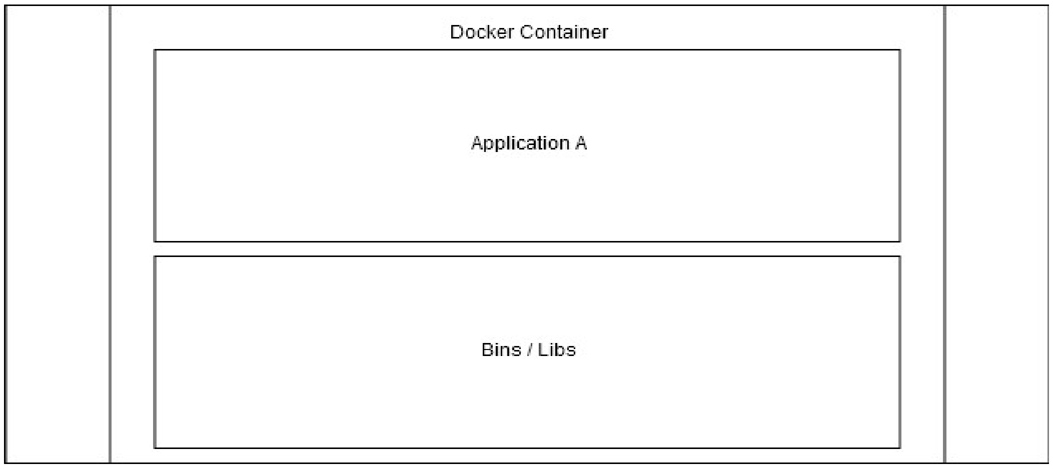


Figure 3. A container consists only of a given application and its dependencies.

. Source: https://devopswithdocker.com/part1/

If we need a project that must be run on a Ubuntu distribution (is an operating system), we can either use a Ubuntu container or a virtual machine with a distribution on it, to set up the virtual machine we will reserve for it a certain amount of storage space in the device, and a space of RAM, , then we will download a complete Ubuntu distribution and install it in the virtual machine Let's try or create the project inside the virtual machine, and this will take a very long time, while our use of the Ubuntu container will not take much time, because the Docker does not install the entire distribution inside the container, but it installs only the distribution packages, and it uses the kernel of the operating system On your device, this will help reduce the use of device resources in addition to the operating speed, as the kernel will be shared between the container and your operating system, and the difference will be only in packages because it works inside the container separately from the operating system. To clarify the difference further, suppose you want to run three different applications on your personal device (or on a server, for example). But each application needs different packages from the other application. These packages may conflict with each other, so you need to isolate these applications from each other. The following image explains how to isolate these applications with virtual machines (for example, by using VirtualBox) and how to isolate them with Docker containers, see figure 4.

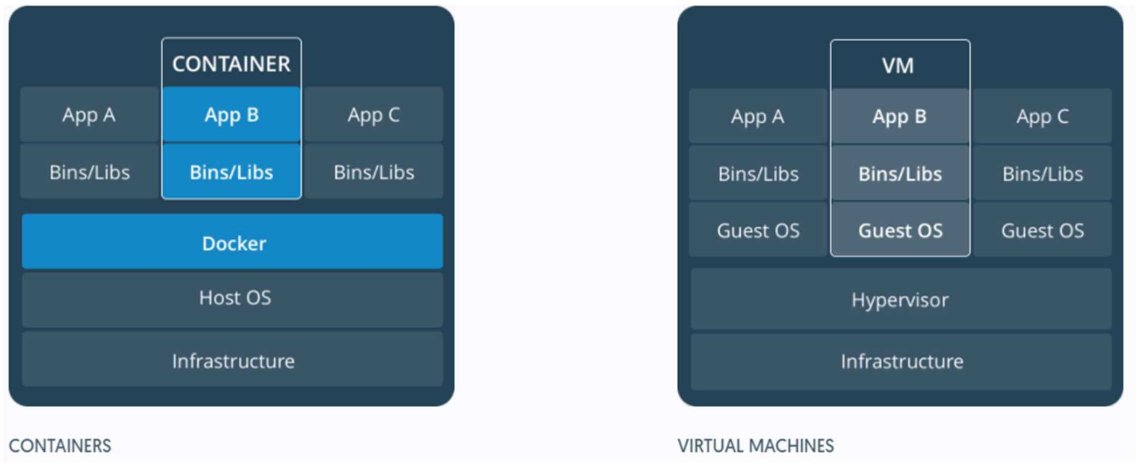


Figure 4. The difference between container and VM.

By the end of this interesting day, I gained so important and much information, which increased my knowledge about Docker and containers. For me, I felt as if I need to know more about this important topic.

2.1.10 Monday 8 June 2020: Creating Containers

An exciting day ahead. My today's goals are:

- How to create containers
- the difference between Docker image and containers.

Both are very important for any researcher and considered essential to learn Docker.
In order to create a container, you need to define the Docker on the attributes of the container that you want to create, for example, if you want to create a container for the Ubuntu system (that is, it contains libraries and packages for Ubuntu) then you need to tell Docker that you need an Ubuntu container with a version of 16.04 for example, and here we need for the mage.
The image is a text file that contains the attributes and settings of the container that you want to create, and you can create more than one container using the same image. The images are built based on other images. If you want to create an image that contains special packages to run, for example, the Apache server. You will often fetch an Ubuntu image or an image of any other system and then write specific commands to install and set up the Apache server. In this way, Apache's image has been created, but you have already re-used an image and added some packages, and edited its settings.
In order to create a reliable image, we need an image of operating systems, which means that we need to have an image source. Docker provides stores where users can download images from (Docker Hub). Some of these images are formal and are provided by Docker Team. Other images are created by a group of developers who uploaded them for instant use. However, users can create their own images if they could not find what they are looking for.

2.1.11 Docker Images vs Containers

"When discussing the difference between images and containers, it isn't fair to contrast them as opposing entities. Both elements are closely **related and are part of a system defined by the Docker platform**.
Images can exist without containers, whereas a container needs to run an image to exist. Therefore, containers are dependent on images and use them to construct a run-time environment and run an application.
The two concepts exist as essential components (or rather phases) in the process of running a Docker container. Having a running container is the final "phase" of that process, indicating it is dependent on previous steps and components. That is why Docker images essentially govern and shape containers".
Source: https://phoenixnap.com/kb/docker-image-vs-container

The day was interesting, where I achieved the objectives I set for today, which helped me to know more about creating containers and the difference between image and container. By the end of these goals, I am ready to start with Docker practically.

2.1.12 Weekly Analysis

During the week, I have been getting familiar with Docker. I addressed different concepts of Docker. Where I addressed the containers in general, the benefits of the container in addition to the cases of using them. Also, this week I went through Docker deeper, where I read about Docker engine, Docker architecture besides the uses of Docker, which are important to know why we need Docker.

I also found that we need to define the concept of Docker, which gave us a general overview of it. The knowing of the difference between the container and the virtual machine was important because I considered it as one of the bases which will help me to increase my knowledge with Docker, in addition to creating containers and the comparison between Docker images and container.

I also talked about my feeling during the study which, reflect my insisting to learn more and more. I think I now have a lot of information that will help me to get started with Docker.

I think that the information that I got is very important and necessary, so I will continue searching for this important topic even after finishing my study. My question after the end of this week is, “what should I learn more about this important topic?” Finally, I think the results I gained during this week was very productive.

2.2 Week 2

2.2.1 Tuesday 9 June 2020: Get started with Docker

Today I will start with Docker practically because I feel it is the time to go deeper and know more. My objectives for today are:

- Get started with Docker by applying the commands of Docker.
- Taking an example of running an Ubuntu container.

I will install Docker on my computer then run Hello world for the first time for me. Afterward, I will run the Ubuntu container.

After installing Docker on your device, execute the following command:

```
docker run hello-world
```

The run command creates a container for the image (in this example, the name of the image is hello-world) and then runs this container. The image hello-world is an example used for testing only; the run command assumes that the image is already on your device. If it does not find the image, it will automatically pull the image from the Docker repository and then create a container for the image.

2.2.2 Example of running an Ubuntu container.

First, to run a container, there must be an image containing a description of the container. The images are fetched from Docker Registry, which is a group of stores that contain Docker images. Of course, you can create your own Registry and provides a Docker Registry with the name Docker hub, as mentioned earlier. From it, we bring the official images and some other images that the developers upload them.
Run the following command to pull out an image from the Registry, which is the Docker Hub by default:

```
docker pull nginx:1.14
```

The pull command is used to drag or download an image to your device. Nginx is the name of the image. The part that falls after the colon sign (:) is a tag that varies with the image. But it represents the program version or the package that this image runs, and you can not write the tag to get the last version. You can see all images available on the Docker Hub website.

Now try the command:

```
docker images
```

It will show you all the images loaded on your device, and now to run the Nginx container execute the run command:

```
docker run --name mytest nginx:1.14
```

The container will now run (if you run the run command and the image is not available, it will execute the pull command automatically). The name command is an optional command that you can implement to name the container, with the name you type after the command. If you do not name the container, then an automatic name will be given to it. To exit from the container, the terminal, press ctrl + d or type exit command. To list all containers that are currently running, run the command:

```
docker ps
```

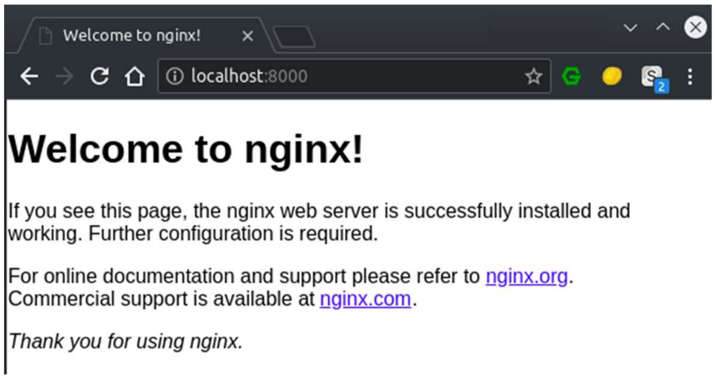
You can also add the -a option to the previous command to see all previously run containers, including containers that are not currently running. If you exit from the terminal that you ran the command through and then executed the ps command, you will notice that the container was closed once the terminal is closed; this is because Docker containers are in one of two states, either working in the front (Foreground) or in the background (Detached). By default, Docker containers work in the foreground, meaning that they work within the terminal emulator that works now, and do not work as a Demon in the background, so We use the d- command to run the container in the background:

```
docker run -d --name mytest nginx:1.14
```

Note: If you implemented the previous command to run the container, an error will occur when executing this command. Because you are trying to create a container that already exists, either you change the name or delete the old container.
Now to check if everything is working properly, we know that Nginx is running on port 80, so you will probably type localhost: 80 in the browser to try it out, and surprisingly it won't work.
Because the container is an environment isolated from your operating system. Nginx actually works on port 80 inside the container, but it does not work outside it, so we use p- with the run command to post ports from the container to the host:

```
docker run -p 8000:80 -d --name mytest nginx:1.14
```

With this command, Port 80 will be published from inside the container to Port 8000 outside, so you can now write localhost: 8000 inside your browser, and as a result, you will see the welcome page from Nginx as shown in figure 5.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 5. welcome page from Nginx.

Source: https://3alam.pro/3mmarg97/series/introduction-to-docker/lessons/what-is-docker

Since the container is running in the background now, it will not stop working as you exit the terminal, and to stop the container from running, use the stop command:

docker stop mytest

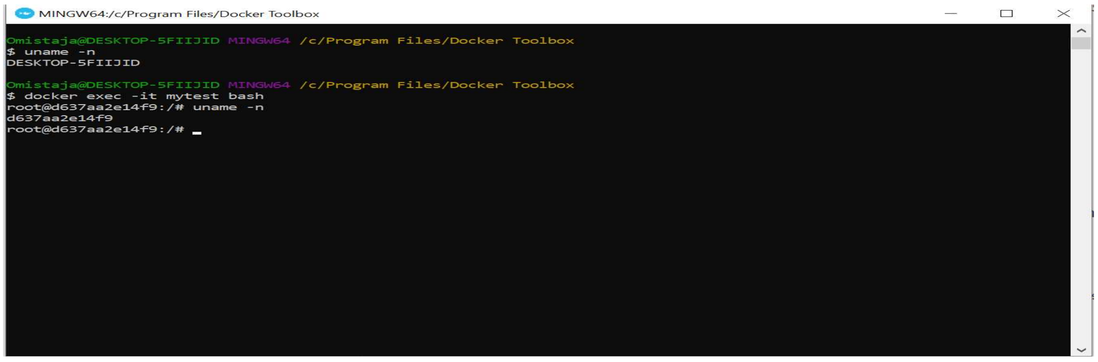
To run a previously created container, use the start command instead of the run command. You do not need to create a container again:

docker start mytest

Note: When using the start command. You can not specify the port or specify whether the container can work in the background or not. But the container will work with the same settings that you specified when you created the container using the run command. When you run the container test, you will be able to access Nginx through port 8000. The container will work in the background.
You can connect with the terminal of the container on which you are working to execute orders using the exec command:
docker exec -it mytest bash

The exec command executes a shell command inside the container, while it -i opens the input stream (STDIN) and t creates (TTY) to deal with the terminal, so the two options together allow you to stay inside the container terminal and execute the commands.

Using the exec command without these two options. The command will be executed without allowing you to enter the terminal to execute the commands. Since the exec command needs to pass a terminal command to it, you can pass the bash to open it and be able to execute the commands inside it, see figure 6 to get more about the idea.



```
Omistaja@DESKTOP-5FIIJID MINGW64 /c/Program Files/Docker Toolbox
$ uname -n
DESKTOP-5FIIJID

Omistaja@DESKTOP-5FIIJID MINGW64 /c/Program Files/Docker Toolbox
$ docker exec -it mytest bash
root@d637aa2e14f9:/# uname -n
d637aa2e14f9
root@d637aa2e14f9:/# _
```

Figure 6. The execution of docker exec command.

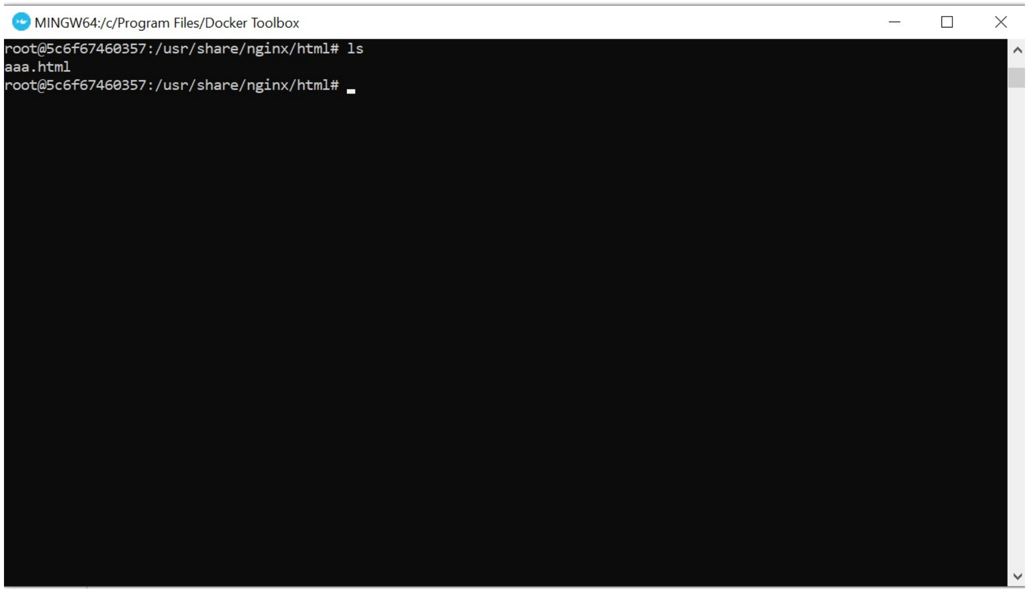
When executing the uname command, my device’s hostname was shown at the beginning, but after executing the exec command you notice that the shell prompt has changed due to the use of the container shell. When executing the uname again, we notice that the hostname has changed, so it is possible to do any command here to apply to the container. If you no longer want the container, you can delete it using the rm command:

docker rm mytest

Finally, one of the great features of Docker is the Volumes. Where you can install (mount) a folder from your device to another path in the container. One of the benefits of this is that you do not need to copy your project files. For example, from your device to the container. But you can also access the same files from within the container or from your device, each in a different path, and to try this feature, I created a file named aaa.html on my device, and I will run a container that can access this file so that Nginx provides me with the file when I enter its path and I can open the file through the browser:

docker run -d -p 8000:80 --name mytest -v ~/Desktop/Workspace:/usr/share/nginx/html nginx

The -v option is to do the mounting, ~ / Desktop / Workspace is a path on my computer that contains the aaa.html file, and usr / share / nginx / html / is the path Nginx uses on the container. See figures 7 and 8, which are showing the result.



```
root@5c6f67460357:/usr/share/nginx/html# ls
aaa.html
root@5c6f67460357:/usr/share/nginx/html#
```

Figure 7. The result after applying the command docker run.

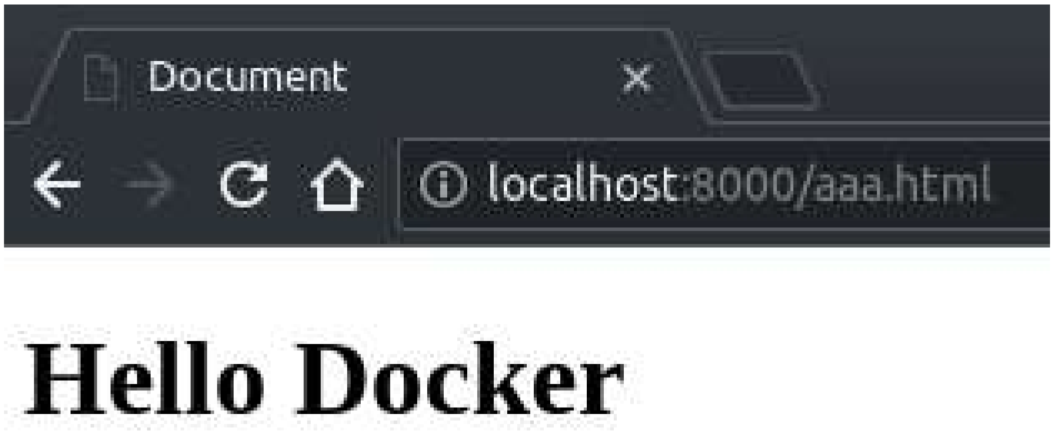


Figure 8. The final result.

What a good day. I addressed a good number of commands of Docker. I achieved my goals of today better than expected. I read and practiced several commands. The quantity of information that I got was very great.

2.2.3 Sunday 14 June 2020: A practical example of creating an image

2.2.3.1 Introduction

We previously explained what a Docker is, clarified its importance, explained the concept of a container, created a container, and explained some of the commands for dealing with the container. Now we will explain how to create an image based on another image.

The container needs vary according to the project you are working on. Packages, requirements, and settings may vary from project to project. You might be working on a project that needs a copy of Python as well as settings for Nginx to use as a proxy. In another project, you may need a specific copy of PHP with a copy and settings for the Apache server, often you will search in the Docker Hub for an image that is right for you, but often you may not find exactly what you need, then you will have to create an image to use to create the container you need.

2.2.3.2 Create an image

The images are created with specific commands written inside a text file called Dockerfile. It contains chain orders or instructions explaining to Docker how to build the image. Docker images are often created based on other images. In order to create, for example, an image of Nginx, you need to build it based on the Ubuntu image because Nginx will not work without an operating system (in a few cases, Docker images may already be created from scratch).

2.2.4 Weekly Analysis

This week I got started with Docker practically. I started by installing Docker on my device. I noticed that if you want to install Docker on Windows, you need to upgrade your windows version to premium. The command "docker run hello-world" was very enough as the beginning of practicing and understanding Docker's commands.

By taking the example of running Ubunto container, I can say I have learned a lot about Docker, at least as a beginner for me, where I applied many and different commands. Going through this example made me feel that the learning of Docker is not difficult but requires working hardly. In my view, it is not different from learning any programming language.

This week's outcome was really impressive, and I think that I am starting to acquire a new skill, which is Docker.

2.3 Week 3

2.3.1 Monday 15 June 2020: Create the image

Another exciting day because I set it to create an image. This day will be practical, totally. I think I will achieve my goal by practicing several commands of Docker and patience because I am sure I will face some difficulties for the first time.
Initially, create a file named "Dockerfile" which is the text file that will contain instructions for creating the image. The Docker commands are written as follows:

INSTRUCTION [arguments]

Where INSTRUCTION is any Docker command, where we will learn some of them now. Arguments are the sentences that will be written after the command as information that is passed to this command to process according to the command. Comments can be written in the file preceded by the hash (#) sign.

The first command that writes at the beginning of all Dockerfile files is the FROM command and passes the name of the image you want to base your image on, for example:

FROM ubuntu:16.04

With this line, we have specified for Docker that we will build an image based on the Ubuntu image with version 16.04. When creating a container from your image, Docker will pull the Ubuntu image and then execute the rest of the commands based on the Ubuntu image.

One of the most frequently used commands is the RUN command, which executes commands in the container terminal, for example:

RUN apt-get update && apt-get install -y nodejs npm
RUN mkdir /application

Note that the sentences after the RUN command are all normal commands. The first updates the repositories and installs the NodeJs package, and the second will create a file. You can type the RUN command multiple times.

If you want to copy a file from your device to the container, you can use the COPY command or the ADD command, and they do almost the same thing, but the COPY is more readable, and the ADD command supports adding files to the container from a specific link and not just from your device. The benefit of these two things is copying your project files to the container or copying a configuration file for a specific package or any other package. **Example:**

```
COPY package.json /application
ADD https://example.com/folder/file.js /application
```

Sometimes you may need to execute a command in the terminal while the container is running and not during its build (with each container run). Here you can use the CMD command and execute the command you want (you can only use one copy of the CMD command, and if you write more than one, the last one will be executed only). Use the WORKDIR command to specify the working folder in which to execute commands, for example**:**

```
WORKDIR /application
CMD npm start
```

Or you can use ENTRYPOINT for the same function. But what is the difference between CMD and ENTRYPOINT?

In short, most images use the command /bin/sh -c as the ENTRYPOINT to it, then the command in the CMD is passed to the ENTRYPOINT, so you can change the ENTRYPOINT, and the command in the CMD will be passed to that ENTRYPOINT

Example:

```
ENTRYPOINT ["/bin/cat"]
CMD ["/path/to/file"]
```

This command will print the file specified in the CMD using the cat command. This path has been passed to ENTRYPOINT.

Sometimes you may need to set the Environment Variables in the image, and you can do this using the ENV command as in the following example:

```
ENV VAR=value
```

You can also specify more than one variable with one ENV command:

```
ENV VAR1=value \
```

VAR2=value

With the EXPOSE command, you can specify the ports used within the container. If we create a server using Node which uses Port 80, we will set the command EXPOSE as follows:

EXPOSE 80

You may be wondering, what is the benefit of specifying the ports used with EXPOSE, and we will always use the -p option under the run command to specify the port?

When specifying the port using EXPOSE, you will not be able to access this port from your device unless you use the p- option. But other containers will be able to access this port within that container, but when you use the p- whether with or without EXPOSE, the result is the same. However, it is preferable to use EXPOSE to act as documentation of the ports to be used. You can also write EXPOSE and use the option -P instead of -p to post all the ports specified in EXPOSE abroad without rewriting them under the run command. To read about EXPOSE in detail, see this answer.

https://stackoverflow.com/questions/22111060/what-is-the-difference-between-expose-and-publish-in-docker/22150099#22150099

For more details on more Docker file commands, read the official reference. After completing writing the previous commands for the file, this is the complete file:

```
“FROM ubuntu:16.04
# Fetch the Ubuntu image
RUN apt-get update && apt-get install -y nodejs npm
# Update repositories and download NODE
RUN mkdir /application
# Create the folder that contains the application
COPY package.json /application
# Copy the npm package file
# To the folder we created
# Package.json file must be available
# Besides the Dockerfile file
WORKDIR /application
#  Select working folder
CMD npm start
# Executing the command npm start”
```

Now we are going to build the image using the command:

docker build -t mytest:latest .

The -t option enables you to specify the name and tag for the image. In the previous example, we chose the name mytest for the image, and we used the latest as the name for the tag, then we specify the folder that contains the Dockerfile file and put a point mark to indicate that the file is in the same working folder that I am browsing. Docker will execute the previous orders line by line and print the line it is working on, and in case of any error, you will know where it happened. After completing building the image, it will print for you:

```
Successfully built a499ec6eafd0
Successfully tagged mytest:latest
```

Of course, your image ID will differ. You can now run the docker images command to display all your images, and the image you created should appear:

Now try running a container from the image we created:

```
docker run mytest
```

The output, of course, will vary depending on your package.json file.

Finally, I can say that I really achieved my goal when I built the image successfully. It took so long time to read and practice. I got a lot of information, which made the feeling that I am progressing day by day.

2.3.2 Friday 19 June 2020:   Use the Docker Compose

Introduction

My objectives for today are to learn how to create several containers by using Docker compose. I need to read about Docker compose to understand the idea. I think it will be a theoretical day.

One of the most important benefits offered by containers is the ability to prepare them easily. Whenever you want, you can prepare a container that simulates any environment quickly, so one of the new practices is to make a single container do only one job. For example, to develop a PHP site, it is better to create a container for the webserver only, a container for databases, and a container for PHP. The greatest benefit in this way is the ability to install or remove any element (container) from this design easily without the need to take a long time in preparing packages and environment variables, and others. For example, if you need to use Redis in your project, all you have to do is fetch an image of Redis and link it to the rest of the containers. If you need to change the database, for example, from Mysql to MongoDB, you must replace a MongoDB container with a Mysql container. All of these operations can be difficult if you are working on one container. Each time, you have to edit a lot of settings in Dockerfile, which may be very large due to the use of a lot of tools in it, and the time is golden for the developer, you will not like, of course, to delay the delivery of a project or delay in launching it. It is better to finish the project in less time than the specified time, and for this, the programmer should use tools that facilitate and speed up the development process. Based on the previous, the Docker Compose tool is a great tool for attaching containers to each other, so by using one file, you will be able to link the containers easily (the method of installing the tool). In this section, we will explain some of the Docker Compose commands, and then we will apply a realistic example to it.

Today I read about Docker compose in addition, I also watched some videos. I think I got good and useful information related to my today's goal.

2.3.3 Saturday 20 June 2020: Get started with docker compose

Yesterday was theoretical, where I read about Docker compose, but the information that I got was quite enough to start with Docker compose. So, my today's goal is to get started with Docker compose.
As with creating an image, Docker Compose relies on a file that contains the instructions it needs to know what to do. Compose instructions are written in the YAML language inside a file as YAML or YML. The first step to using Docker Compose is to create a file named docker-compose.yml.

At the beginning of each compose file, we specify the version of the file that we are working on. Each copy contains different advantages from the previous version. You can see the differences between copies of the compose files and the Docker compose versions that run each version of it. Refer to this link, and in this explanation, we will adopt version 3.

With Docker Compose, you will be able to distribute each part of your application on several containers. So, you can run NodeJs on one container, Nginx on another container, and a third container to deal with your project files, each of these parts is called Service, open a new Code block inside services to define each Service you have:

```
“version: "3"
services:
  mysystem:
    image: ubuntu:16.04
  anotherone:
    build: ./myimage”
```

In this example, we have two services, the first one is called mysystem, and it pulls the ubuntu image from the Docker Hub. The second one is anotherone, and instead of fetching the image from the Docker Hub, it builds the Dockerfile file located inside the myimage folder. This, of course facilitates the running of these containers. In large projects, when dividing the project requirements into services, you will need to run a large number of containers, and Docker Compose will facilitate you to run or stop them all at once using one command.

We got to know the image and build command, but of course, there are many other commands related to services, such as the ports command that helps you define and direct ports (in the Dockerfile files, the EXPOSE command or the use of -p is used after the Docker command to specify the ports, but dealing with the ports in the Docker-compose file Easier).

```
“version: "3"
services:
  mysystem:
    image: ubuntu:16.04
  anotherone:
    build: ./myimage
    ports:
      - 8000:80
      - 9000:9000”
```

You also notice that we specify the port on which a specific service is running in the container (port 80, for example) and direct it to a specific port on the hosting device (port 8000 in the example). Using port 8000, by typing localhost: 8000 in the browser, for example.

To connect between containers, Docker Compose automatically creates a network between the containers so that they can be connected, you can, of course create your own networks, but we will not address this issue in this simple series.
To ensure successful communication between containers, the depends_on command is used. Where you specify the services (containers) on which the service you used the depends_on command depends on and what the Docker Compose does is sorting the work of the containers, as the running of a container should not relying on another container that has not been created.

```
“version: "3"
services:
  mysystem:
    image: ubuntu:16.04
    environment:
      MY_VARIABLE: 1342
    depends_on:
        - anotherone
        - third_one
  anotherone:
    build: ./myimage
    ports:
      - 8000:80
      - 9000:9000
  third_one:
    image: example”
```

Note the use of depends_on on the first container (mysystem), thus Docker Compose will ensure that the mysystem container will be run before anotherone and third_one, so that the container playback order is as follows:

mysystem -> anotherone -> third_one

Or

mysystem -> third_one -> anotherone

A useful command is also the container_name command, and by using it, you can specify the name of the container, and it makes the name-- option in the Docker run command. The service name (such as mysystem or anotherone) does not represent

```
“version: "3"
services:
  mysystem:
    image: ubuntu:16.04
    container_name: mytest”
```

the container name, but rather a name used to deal with the service within the Docker-compose file mostly the container name will have the same name as a serial number, but to specify a specific name for the container, you will need the container_name command.

Finally, we will touch on a very important feature, which is the Volumes, and we have explained it beforehand, but we give an example of how to use it in the Docker Compose file:

```
version: "3"
services:
  mysystem:
    image: ubuntu:16.04
  volumes:
   - ./myfolder:/var/www
```

To run the services you defined in the Docker-compose file, in the folder containing the file, execute the command:

docker-compose up

You can use -d as the flag for the previous command to run services in the background.

To stop services:

docker-compose stop

As for stopping (and deleting) all services (containers), networks, and Volumes that were created with the up command, we use the following command:

docker-compose up

Sometimes you use a Dockerfile file and need to rebuild it after editing it, and then you do one of the following:

docker-compose build
docker-compose up –build

The first command rebuilds the images, the second rebuilds and runs them.
Although the features we covered are just a few of the advantages of Docker Compose, and there are many other commands that are detailed. For more, see the official documentation. Generally, the competence gained during the day was very good.

2.3.4 Weekly Analysis

This week was really eye-opening. Where I read and learned more about Docker compose. I touched on Docker compose in general. Then I got started with it practically. Working with Docker compose was very interesting because it deals with several containers.
I addressed many of the Docker compose commands, which were very useful. Where the examples were clarifying the idea behind Docker compose and how it works. I think I touched on the most important commands in this week related to Docker compose. In general, I think I can say that I summed up the most important things you need to know about Docker compose.
Although I learned much about Docker, I still feel that I need to learn more to develop my skills with Docker. I found that the importance of Docker increasing day by day. In my view, any company or programmer needs to learn this skill because it will become a basic thing in the future.

2.4 Week 4

2.4.1 Wednesday 1 July 2020: Introduction to Robot Framework

Robot Framework is also a new technique for me. So, I need to study this concept in general to get a comprehensive idea about it. That will be my objective I set for today. Testing is important for any activity to be succeeded. This should be for software also. It covers the verification and validation of the software. Testing practices include the unit and functional testing that proves the needed functionality. However, the testers should make manual efforts to make sure that the program that should be tested works as required, as part of the software acceptance process. Because of the increase in the size of the programs, the process of the testing became automated, leaving behind huge information in the form of audit trails of tests that have to be available to the user at a glimpse and without which the purpose of the acceptance test automation is defeated.
Robot Framework adds this missed link that combines with the testing tool used for implementing acceptance testing over the software and builds this implementation into separated structures of tests sorted in an appropriate hierarchical way. This is led to correct execution and generates orderly reports. Because of an open-source

framework, this tool enables you to perform acceptance testing in a hierarchical way and generating automated reports. In addition to it allows you to build your own tool. Or even use it with other tools as part of a custom test environment setup required for your team.

It was an easy day. I managed to achieve my goal by reading different articles. The information I got was useful and broadened my horizon to advance my understanding of this important topic.

2.4.2 Thursday 2 July 2020: Robot Framework definition

Yesterday I studied Robot Framework in general; therefore, I found that it is important to define it more accurately. Today's objective will be Robot Framework definition especially.

Robot Framework is an open-source automated framework. It can be used for automated testing and robotic process automation (RPA). Many companies are using Robot Framework in their software development. It is open and extensible; we can merge it with another tool to get powerful automation solutions. Because it is open-source, it is free to utilize without any license costs.

Robot Framework has easy syntax, using keywords that a human can read. It is possible to extend its abilities by libraries of Python or Java.

Robot Framework is hosted on GitHub. You can find more information, documentation, source code, and issue solutions there. Downloads are hosted at PyPI. It is an operating system and an independent application. The main framework is executed by Python and runs on Jython (JVM) and IronPython (.NET).

Robot Framework is issued under Apache License 2.0, and most of the libraries and tools in its system are also open source. The framework, in the beginning, was developed at Nokia Networks and became open-sourced in 2008. Robot Framework architecture is presented in figure 9.

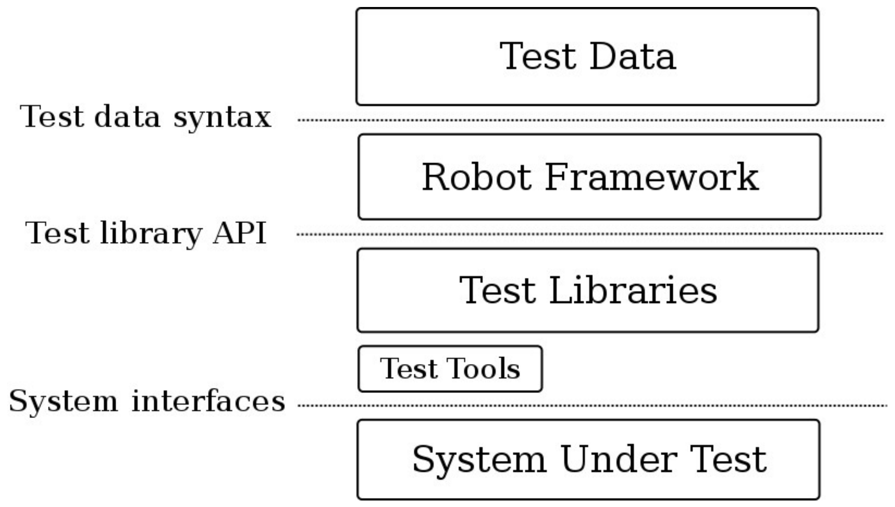


Figure 9. Robot Framework architecture.

Source: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

I can say that I achieved better than expected for the day and got a lot of important information related to the Robot Framework.

2.4.3 Friday 3 July 2020: The importance of Robot Framework

To like something, you must know its importance. From this point of view, I devoted this day to studying its importance of the Robot Framework.

- Robot Framework is easy to use the syntax of tabular for creating test cases.
- It has the ability to create reusable higher-level keywords from the existing keywords.
- Provides clear reports and logs in HTML format.
- It is application independent.
- Provides a simple library API to create test libraries that can be executed with Python or Java.
- Robot Framework provides a command line and XML based output files for continuous integration systems.
- Provides support for Selenium library, which is used for web testing, Java GUI testing, SSH, and so on.
- Creates data-driven test cases.
- Support for variables, for testing in various environments.

The information I got was not very great, but it was important, and by the end of the day, I can say that I successfully achieved the goal that I set for this day.

2.4.4 Weekly Analysis

This week was interesting. Where I completed reading about Docker and starting the most important part of my research, which is Robot Framework; from my point of view, learning Robot Framework is like learning a programming language, and then will increase the skills that I have acquired, which will help me find a job. I considered this week as a start point to go far with this awesome technique. I studied this concept in general; then I touched on its importance. Besides, I saw some small courses about it. I think it was quite enough to start learning.

2.5 Week 5

2.5.1 Saturday 4 July 2020: The Robot Framework ecosystem

I decided that my goal for today is to continue the theoretical side of studying the Robot Framework, so it is very important to look at the Robot Framework ecosystem. The following chart shows a general overview of the framework and various related components, see figure 10.

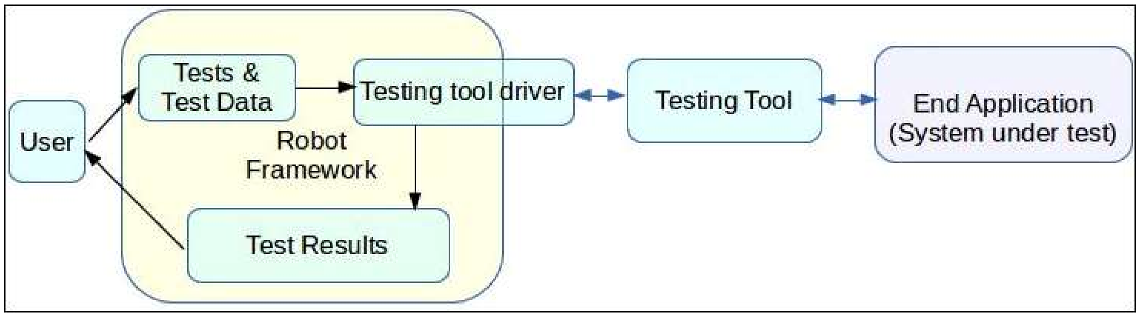


Figure 10. General overview of the framework.

Source:
https://subscription.packtpub.com/book/application_development/9781783283033/1/ch01lvl1sec10/the-robot-framework-ecosystem

Tests and Test Data: This is the tests' configuration, consist of test and data files and folders besides the contents of those that dictate the implementation of the test.

Test Results: These are the results of the tests which are used to determine the results of tests, as well as logs that can be used to evaluate different parts of the test.

Robot Framework: This is the main framework that implements the most important things, to get things done.

Test tool driver: This makes the communication available between the framework and actual tools. It can be specially designed to meet specific requirements with the existing test tool.

Testing Tool: This is the actual software responsible for performing acceptance testing.

End Application (System under test): This is the software we want to test for use by the client or end-user.

Finally, the outcome of the information I got was very great as I studied all the parts mentioned in figure 13 in detail, which helped me get a wonderful understanding of the topic.

2.5.2 Monday 6 July 2020: A small exercise

Today's goal will be practical; as usual, I will apply a small exercise to clarify the idea. To explain the idea of the Robot Framework, I will create a simple example as follows:

1. Create a folder and name it “my first exercise” for example, this folder will be the root folder.
2. Inside my first exercise folder, create a folder and name it testcases, this will contain the test file.
3. Inside testcases folder, create a file and name it TestCase1.robot; write the following inside:

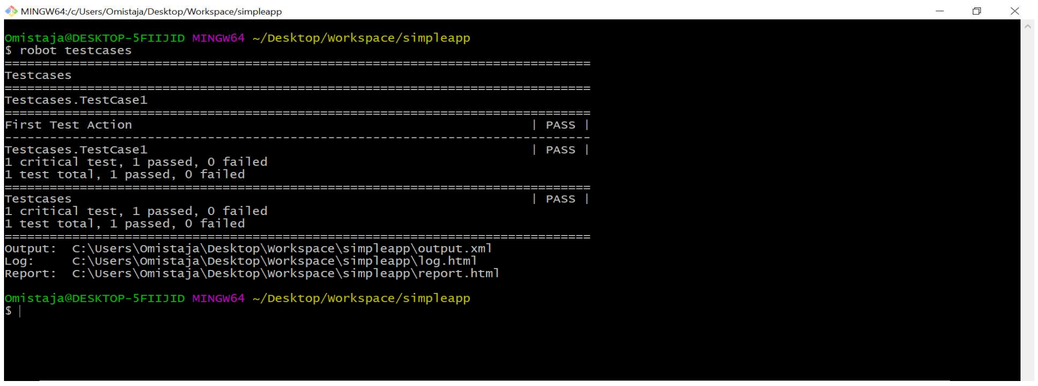
```
***Test Cases***

First Test Action  log  this is a basic test
```

Note that we have to left two spaces before and after the log keyword.

4. Now open the command line and go to the project directory, then run the project by writing the following command:

robot testcases

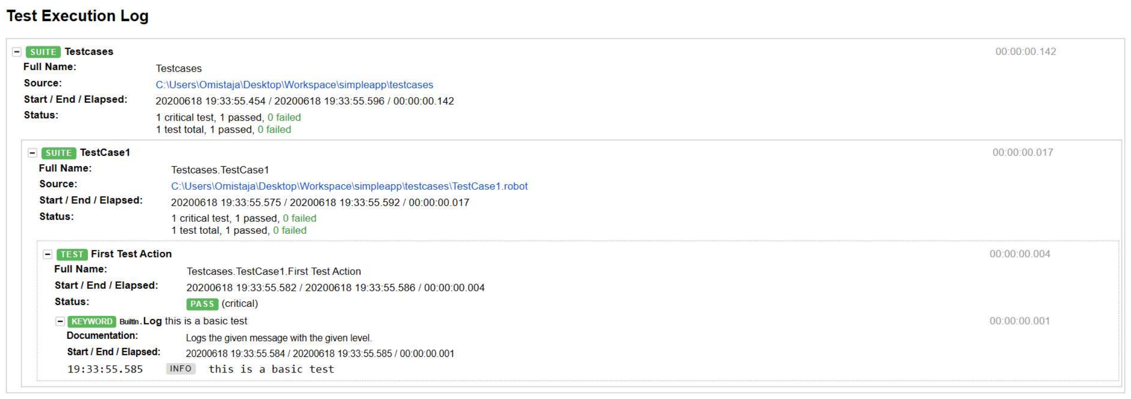
You have to get like figure 11.



```
MINGW64:/c/Users/Omistaja/Desktop/Workspace/simpleapp

Omistaja@DESKTOP-5FIIJID MINGW64 ~/Desktop/workspace/simpleapp
$ robot testcases
==============================================================================
Testcases
==============================================================================
Testcases.TestCase1
==============================================================================
First Test Action                                                     | PASS |
------------------------------------------------------------------------------
Testcases.TestCase1                                                   | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Testcases                                                             | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\Omistaja\Desktop\Workspace\simpleapp\output.xml
Log:     C:\Users\Omistaja\Desktop\Workspace\simpleapp\log.html
Report:  C:\Users\Omistaja\Desktop\Workspace\simpleapp\report.html

Omistaja@DESKTOP-5FIIJID MINGW64 ~/Desktop/workspace/simpleapp
$ |
```

Figure 11. The result of executing testcases.

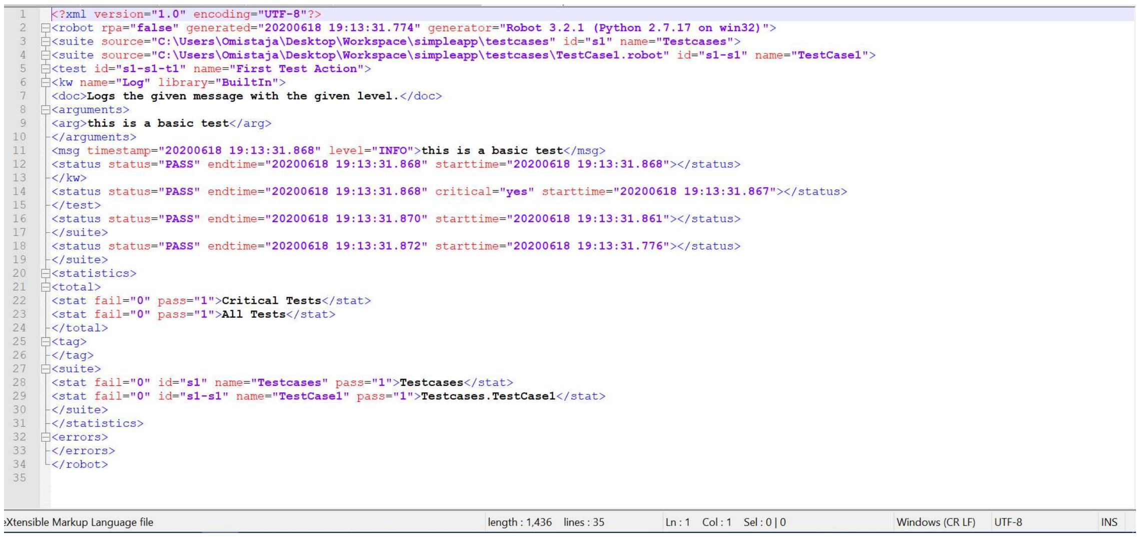
This means that the project ran successfully. You can see the results and log from HTML pages that have been generated from running the project. When we used a simple log statement, we get a logged message in the out log, as shown in figure 12.



```
Test Execution Log

− SUITE Testcases                                                              00:00:00.142
  Full Name:            Testcases
  Source:               C:\Users\Omistaja\Desktop\Workspace\simpleapp\testcases
  Start / End / Elapsed: 20200618 19:33:55.454 / 20200618 19:33:55.596 / 00:00:00.142
  Status:               1 critical test, 1 passed, 0 failed
                        1 test total, 1 passed, 0 failed

  − SUITE TestCase1                                                            00:00:00.017
    Full Name:            Testcases.TestCase1
    Source:               C:\Users\Omistaja\Desktop\Workspace\simpleapp\testcases\TestCase1.robot
    Start / End / Elapsed: 20200618 19:33:55.575 / 20200618 19:33:55.592 / 00:00:00.017
    Status:               1 critical test, 1 passed, 0 failed
                          1 test total, 1 passed, 0 failed

    − TEST First Test Action                                                   00:00:00.004
      Full Name:            Testcases.TestCase1.First Test Action
      Start / End / Elapsed: 20200618 19:33:55.582 / 20200618 19:33:55.586 / 00:00:00.004
      Status:               PASS (critical)
      − KEYWORD BuiltIn.Log this is a basic test                               00:00:00.001
        Documentation:        Logs the given message with the given level.
        Start / End / Elapsed: 20200618 19:33:55.584 / 20200618 19:33:55.585 / 00:00:00.001
        19:33:55.585    INFO    this is a basic test
```

Figure 12. The log from.

You can also easily read the generated XML file, which can be used by another external tool as an input parameter. See figure 13.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<robot rpa="false" generated="20200618 19:13:31.774" generator="Robot 3.2.1 (Python 2.7.17 on win32)">
<suite source="C:\Users\Omistaja\Desktop\Workspace\simpleapp\testcases" id="s1" name="Testcases">
<suite source="C:\Users\Omistaja\Desktop\Workspace\simpleapp\testcases\TestCase1.robot" id="s1-s1" name="TestCase1">
<test id="s1-s1-t1" name="First Test Action">
<kw name="Log" library="BuiltIn">
<doc>Logs the given message with the given level.</doc>
<arguments>
<arg>this is a basic test</arg>
</arguments>
<msg timestamp="20200618 19:13:31.868" level="INFO">this is a basic test</msg>
<status status="PASS" endtime="20200618 19:13:31.868" starttime="20200618 19:13:31.868"></status>
</kw>
<status status="PASS" endtime="20200618 19:13:31.868" critical="yes" starttime="20200618 19:13:31.867"></status>
</test>
<status status="PASS" endtime="20200618 19:13:31.870" starttime="20200618 19:13:31.861"></status>
</suite>
<status status="PASS" endtime="20200618 19:13:31.872" starttime="20200618 19:13:31.776"></status>
</suite>
<statistics>
<total>
<stat fail="0" pass="1">Critical Tests</stat>
<stat fail="0" pass="1">All Tests</stat>
</total>
<tag>
</tag>
<suite>
<stat fail="0" id="s1" name="Testcases" pass="1">Testcases</stat>
<stat fail="0" id="s1-s1" name="TestCase1" pass="1">Testcases.TestCase1</stat>
</suite>
</statistics>
<errors>
</errors>
</robot>
```

eXtensible Markup Language file
length : 1,436 lines : 35
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

Figure 13. The XML file.

Really it was a good day. I achieved my goal easily and understood the main idea behind the idea of the Robot Framework. Generally speaking, I got a fair amount of information.

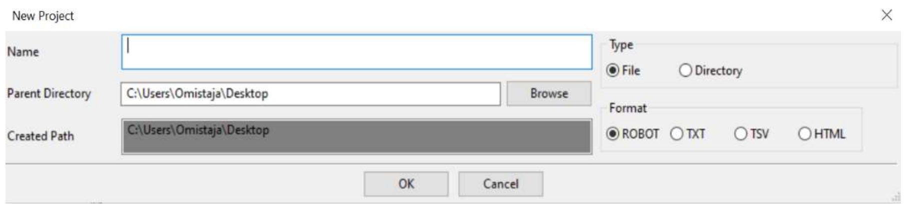
2.5.3 Tuesday 7 July 2020: Introduction to Ride

Today will be a continuation of the practical application of the Robot Framework. I have decided to address the Ride editor for today or in this section because I will use the Robot in GitLab using YMAL file next weeks.

Ride is the editor of Robot Framework; also, we will write test cases in Ride. In this section, we will go through the Ride editor to explain some important features and options that are available in this editor.

Create a New Project

To create a new project, open Ride editor, click on the file menu and select new project. Once we selected this option, we will get the following window, as shown in figure 14. (Robot Framework, July 2020).



New Project

Name:

Parent Directory: C:\Users\Omistaja\Desktop

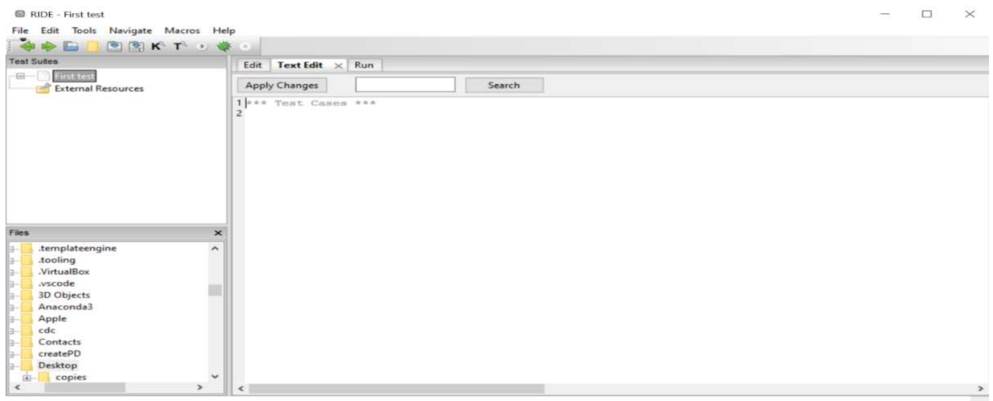
Created Path: C:\Users\Omistaja\Desktop

Type: [✓ File] Directory

Format: [✓ ROBOT] TXT TSV HTML

Figure 14. The window of creating new project.

In the field of name, we will enter the name of the project. Then we will select the path of the project directory in the field of Parent Directory. We could change the location if that required. We can save the project as a file or directory; further, we can save the project in format ROBOT, TXT, TSV, or HTML. Now we will write the project's name and let's name it "First test," as shown in figure 15. (Robot Framework, July 2020).



```
*** Test Cases ***
```

Figure 15. Creating the test cases file.

The project's name appears on the left side, and on the right side, we can see three tabs Edit, TextEdit, and Run. When you click on Edit, you will find that Edit has many options on the UI. In this section, we can add data on which our test is dependent. We can import Library, Resource, Variables, add scalar, add list, add dict, and Add Metadata. The details which are added in the Edit section will appear in the next tab, Text Edit. You can write your code in the text edit section. If there any change happened in the Textedit, it will appear in the Edit section. So, both sections Edit and TextEdit, are dependent on each other, and all changes will be seen on both. We can use the tab Run when the test cases are ready, see figure 16. (Robot Framework, July 2020).

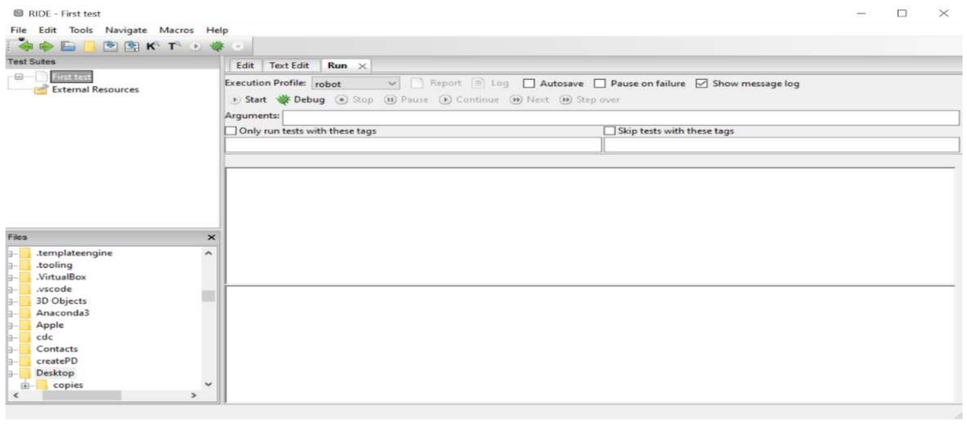


Figure 16. The tab Run.

With Run UI allows to run the test case, and we will see options like start, stop, pause, continue, next test cases, step over, etc... You can also create Reports, Log for the test cases that you are implementing. To create a test case, right-click on the project name "First Test" click on the new test case, then write the name of the test case in the field of name, for example, "TESTCASE0" and press "OK". Once you click "OK" the following you will get like figure 17. (Robot Framework, July 2020).

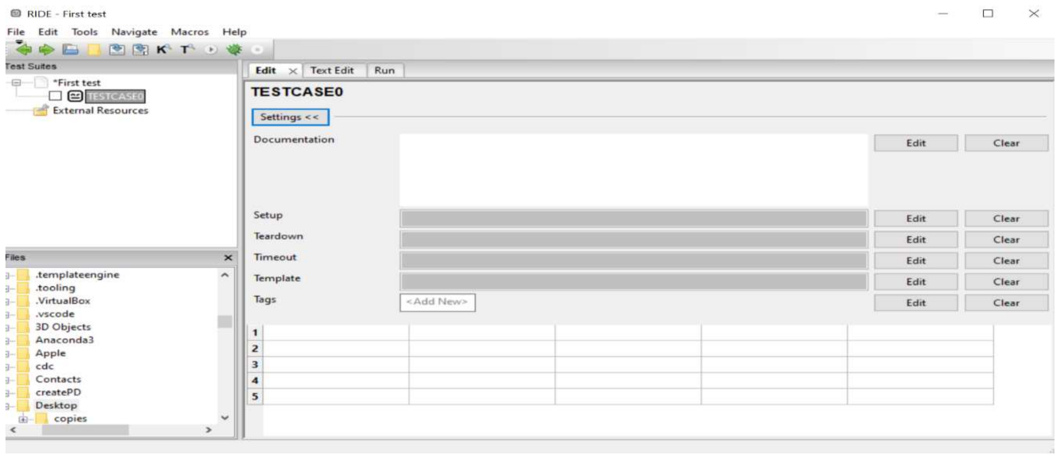


Figure 17. Created test cases file.

The test case contains options like Documentation, setup, teardown, tags, timeout, and Template. They have an edit button; when you click on it, a screen appears where you can enter the details for each option. We can write the test cases in the tabular format, as shown below. Robot Framework test cases are keyword dependent, and we can write the keywords by using built-in keywords or keywords imported from the library. Further, we can create user-defined keywords or variables, etc., in the Robot Framework. You can run or stop the test case from the navigation bar, as shown in figure 18.

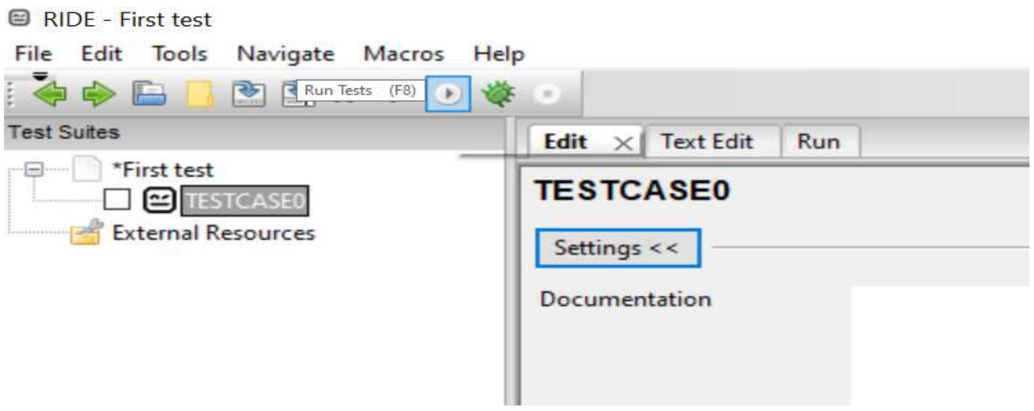


Figure 18. The navigation bar.

You can find the search keyword in the navigation bar like figure 19.

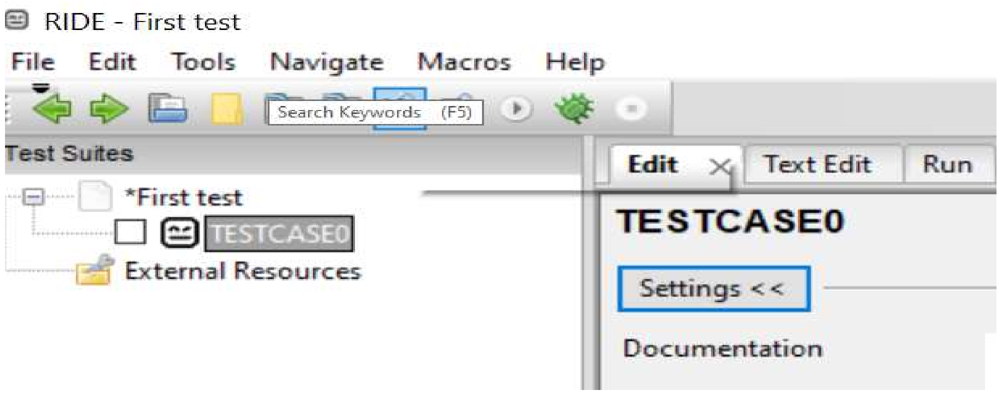


Figure 19. The search keyword.

If you want to get the list of keywords the Robot Framework provides, press ctrl+space in the tabular. It will show you all the keywords available, as shown in figure 20. (Robot Framework, July 2020).

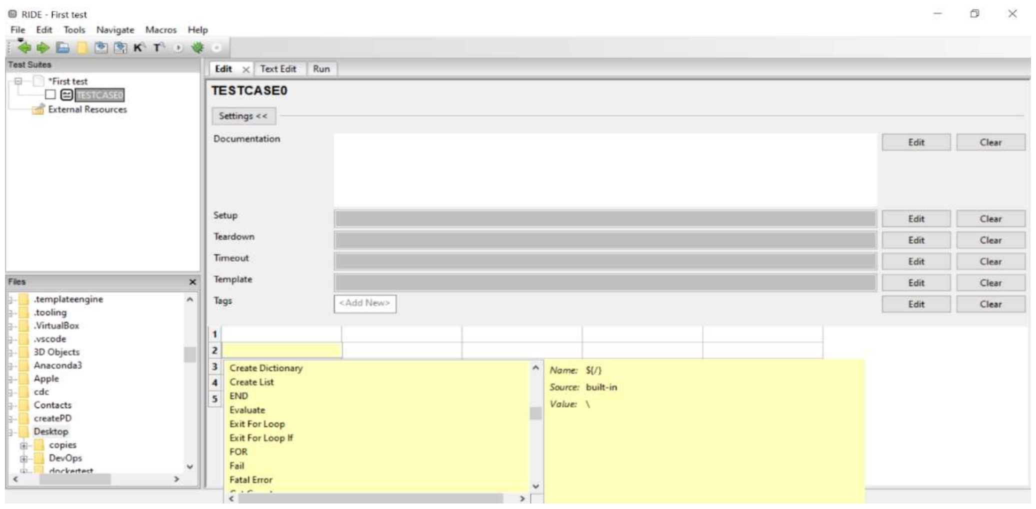


Figure 20. The list of the keywords.

This will help to remember the keyword that we forget. We have the details with each keyword; these details explain how to use this keyword.

All the information that I gained today was totally new and very important for any researcher to know about it. I think I am on the right way to learn Robot Framework. By achieving today's goal, I feel that my level became good, and the coming days will be very practical.

2.5.4 Weekly Analysis

This week went very well and better than expected. I got a lot of valuable information besides the interesting feelings that I felt during the search and practice. I am happy with what I have learned so far.

I studied the Robot Framework ecosystem in detail, which is one of the most important parts that we should learn to understand the idea of Robot. I also took a small exercise which helps the new researcher to get the idea about the Robot where taking a practical example help to understand the idea behind any new topic.

In addition, I had to take a look and use the Ride editor because I will use the Robot within a YML file coming weeks, and this will be useful to get a general idea about Ride and how to practice it using YML file.

2.6 Week 6

2.6.1 Friday 10 July 2020: First Test Case Using Ride

This week will be an extension of the practical side of the Robot Framework. Today I will explain the most important features of this Editor.

In this example, we will explore the RIDE and work on our first test case. First, we need to create a project by opening the Ride editor; then, we should click on the new project from the file menu. Here I am going to name the project “First test” afterward, we will right-click on the project name and click on new test case, I will name it “TESTCASE0” and click Ok. See figure 21. (Robot Framework, July 2020).

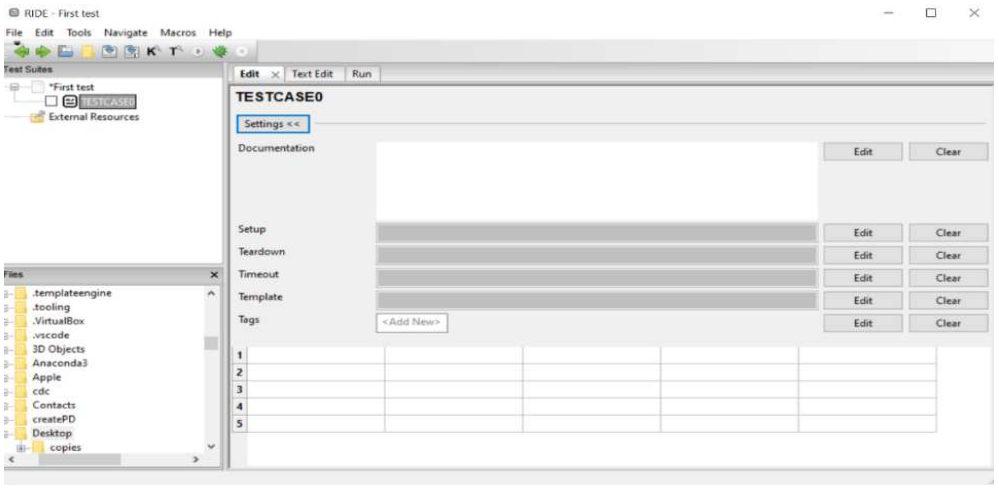


Figure 21. The result after creating test case file.

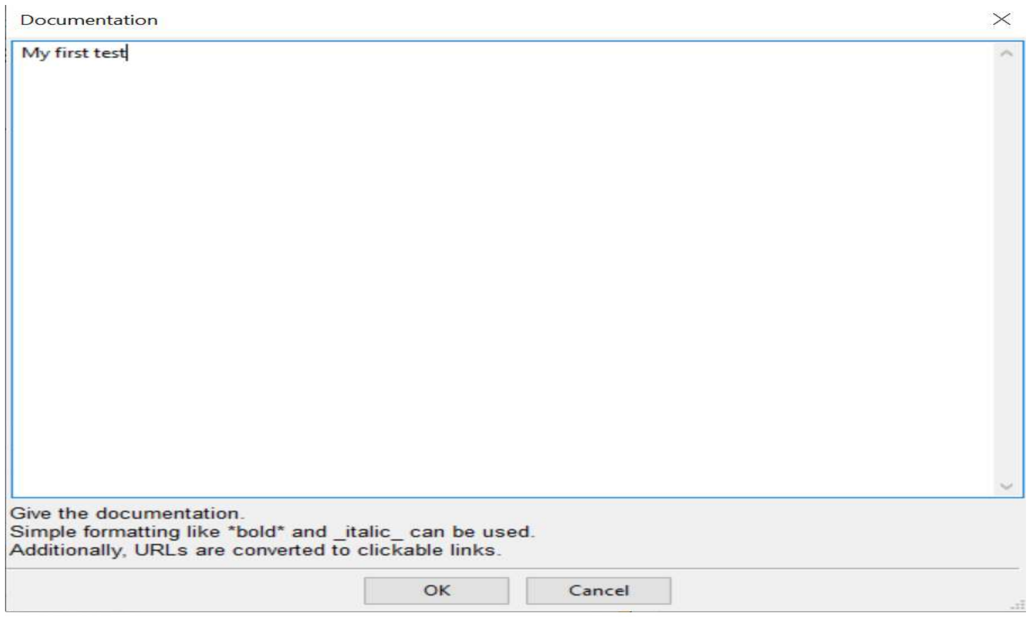
You can see three tabs shown for the test case created: Edit, Text Edit, and Run. The Edit tab has two formats: Settings and Tabular. We will discuss them later.

The Settings Format

We will see documentation, setup, teardown, tags, timeout, and template when we click on Settings.

Documentation

Here you can add details about your test case so that it becomes easy for future reference. See figure 22. (Robot Framework, July 2020).



My first test

Give the documentation.
Simple formatting like *bold* and _italic_ can be used.
Additionally, URLs are converted to clickable links.

Figure 22. The documentation window.

Click the OK button to save the changes.

Setup and Teardown

When we have a setup specific to a test case, it will be implemented before the test case execution, and the test setup that will be implemented after the test case is done for teardown. (Robot Framework, July 2020).

Tags

They are used for tagging test cases to include and exclude specific test cases. (Robot Framework, July 2020).

Timeout

It used to set the timeout on the test suit. We will keep it empty now. (Robot Framework, July 2020).

Finally, I can say that I explain some features of this editor, and I think I still need to go through the features of this editor.

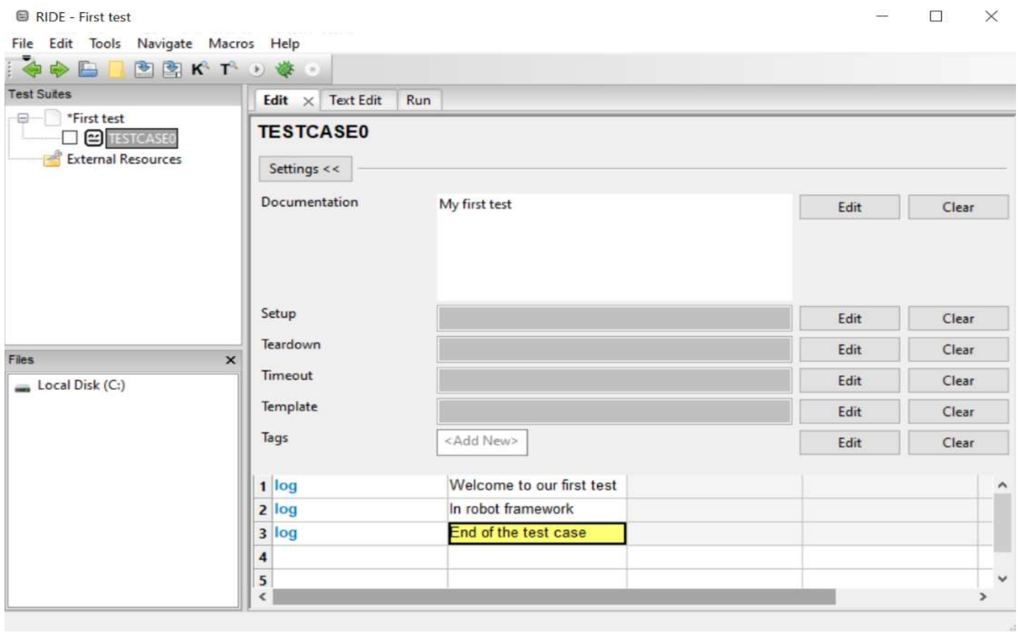
2.6.2 Saturday 11 July 2020: Template

Yesterday went very well. Today, I will continue explaining the features of the Ride editor and taking a practical example.

This will contain the keywords to be used for the test case. It is used for data driven test case. The high-level user-defined keyword is identified in the template, and test cases are used to pass data to the keyword.

We will write our first test cases and implement the same to see the output. In the tabular format.

In this test case, we are going to add some logs, and we will look at the output of it. See figure 23. (Robot Framework, July 2020).



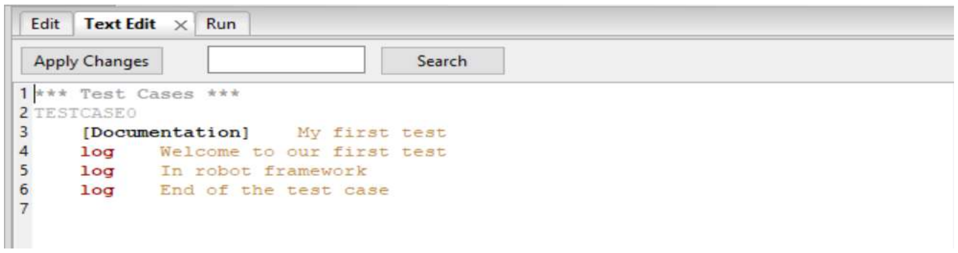
TESTCASE0

Documentation	My first test	Edit	Clear
Setup		Edit	Clear
Teardown		Edit	Clear
Timeout		Edit	Clear
Template		Edit	Clear
Tags	<Add New>	Edit	Clear

1	log	Welcome to our first test
2	log	In robot framework
3	log	End of the test case
4		
5		

Figure 23. Logs we added.

As shown above, we have used the keyword *Log* to log messages. Depended on the keywords specified in Edit, we will get the code in Text Edit, as shown in figure 24. (Robot Framework, July 2020).



```
*** Test Cases ***
TESTCASE0
    [Documentation]    My first test
    log    Welcome to our first test
    log    In robot framework
    log    End of the test case
```

Figure 24. The code in the Text Edit.

You may write the test case in the Text Edit, and you will see the changes in the tabular format as well. Now, I will run the test case and see the output. To run the test case, we need to click on Start in the tab Run, as shown in figure 25.

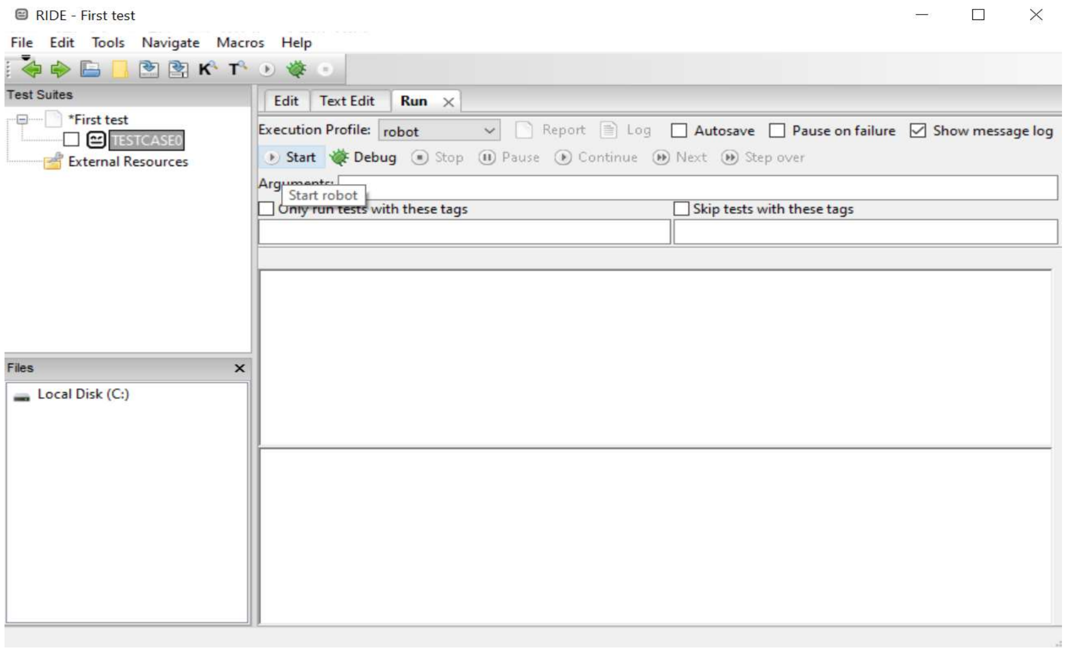


Figure 25. The start button to run the test case.

Below is the output of the test case:

Our test has implemented successfully, and the details are, as shown above, give the status as PASS.

Besides, we can see the details of the execution in Report and Log as screenshot shown in figure 26. (Robot Framework, July 2020).

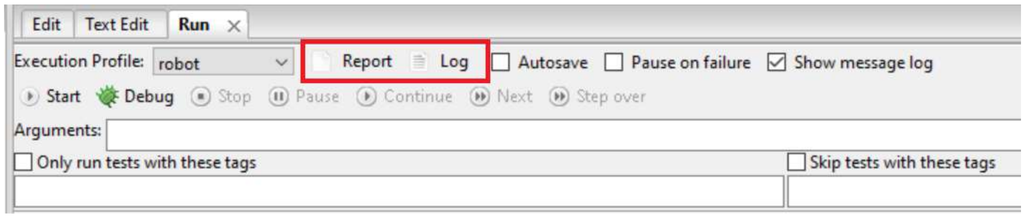
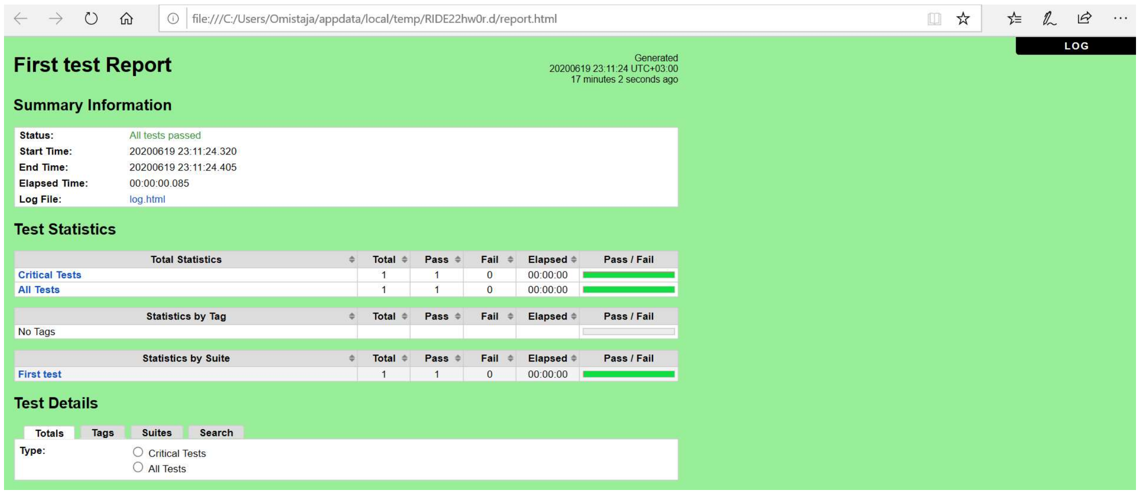


Figure 26. The report and log which are showing the details of the execution.

When you click on the report button, you will see the output like figure 27.


file:///C:/Users/Omistaja/appdata/local/temp/RIDE22hw0r.d/report.html

LOG

First test Report

Generated
20200619 23:11:24 UTC+03:00
17 minutes 2 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20200619 23:11:24.320
End Time:	20200619 23:11:24.405
Elapsed Time:	00:00:00.085
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:00	🟩
All Tests	1	1	0	00:00:00	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
First test	1	1	0	00:00:00	🟩

Test Details

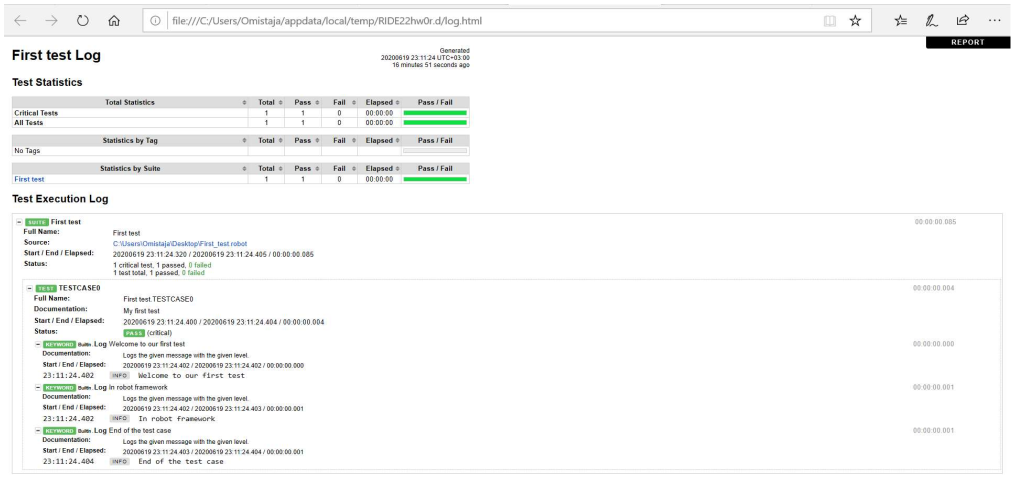
Totals Tags Suites Search

Type: ○ Critical Tests ○ All Tests

Figure 27. The report.

In Report, it shows the details like the start time, end time, path to the log file, status of the test case, etc.

Click on Log; here are the details of the log file. See figure 28. (Robot Framework, July 2020).


file:///C:/Users/Omistaja/appdata/local/temp/RIDE22hw0r.d/log.html

First test Log

```
Generated
20200619 23:11:24 UTC+03:00
16 minutes 51 seconds ago
```

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:00	🟩
All Tests	1	1	0	00:00:00	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

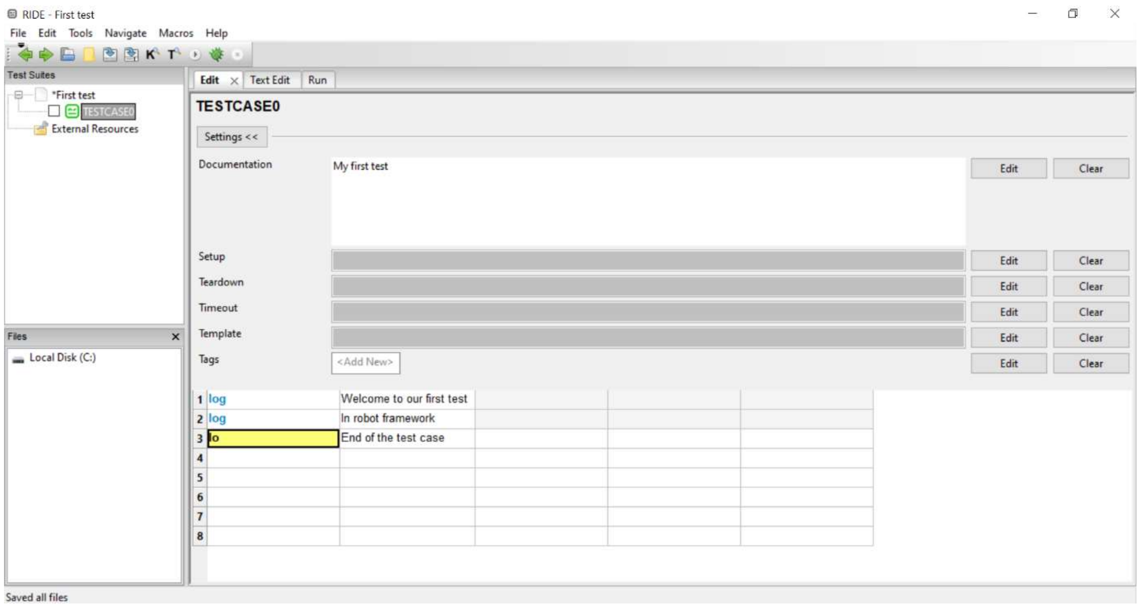
Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
First test	1	1	0	00:00:00	🟩

Test Execution Log

```
SUITE First test                                                                          00:00:00.085
  Full Name:            First test
  Source:               C:\Users\Omistaja\Desktop\First_test.robot
  Start / End / Elapsed: 20200619 23:11:24.320 / 20200619 23:11:24.405 / 00:00:00.085
  Status:               1 critical test, 1 passed, 0 failed
                        1 test total, 1 passed, 0 failed
  TEST TESTCASE0                                                                          00:00:00.004
    Full Name:            First test.TESTCASE0
    Documentation:        My first test
    Start / End / Elapsed: 20200619 23:11:24.400 / 20200619 23:11:24.404 / 00:00:00.004
    Status:               PASS (critical)
    KEYWORD BuiltIn.Log Welcome to our first test                                         00:00:00.000
      Documentation:        Logs the given message with the given level.
      Start / End / Elapsed: 20200619 23:11:24.402 / 20200619 23:11:24.402 / 00:00:00.000
      23:11:24.402   INFO   Welcome to our first test
    KEYWORD BuiltIn.Log In robot framework                                                00:00:00.001
      Documentation:        Logs the given message with the given level.
      Start / End / Elapsed: 20200619 23:11:24.402 / 20200619 23:11:24.403 / 00:00:00.001
      23:11:24.402   INFO   In robot framework
    KEYWORD BuiltIn.Log End of the test case                                              00:00:00.001
      Documentation:        Logs the given message with the given level.
      Start / End / Elapsed: 20200619 23:11:24.403 / 20200619 23:11:24.404 / 00:00:00.001
      23:11:24.404   INFO   End of the test case
```

Figure 28. The log file.

The Log file shows the details of the test execution and keywords we gave for the test case. In both files report and log, we get green color for the status. Let us make changes that will lead us to the failure of the test case fail and see the output, as shown in figure 29. (Robot Framework, July 2020).



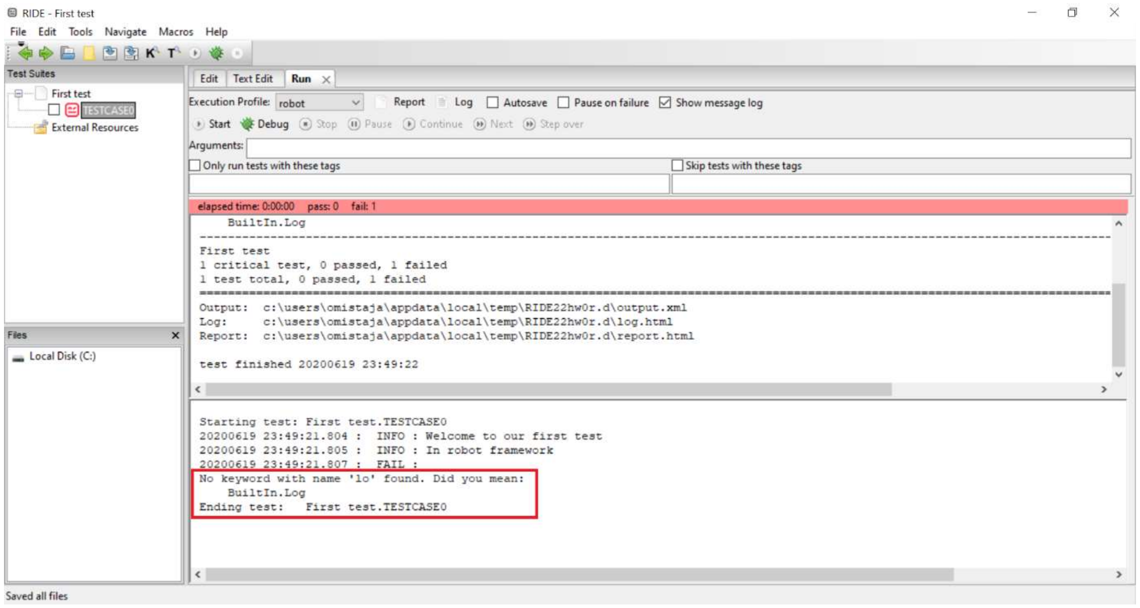
TESTCASE0

Documentation	My first test
Setup	
Teardown	
Timeout	
Template	
Tags	

1	log	Welcome to our first test			
2	log	In robot framework			
3	lo	End of the test case			
4					
5					
6					
7					
8					

Figure 29. The changes we made to create failure case.

In the above test suit, we have written the Log keyword wrongly; when running the test case, it will fail, as shown in figure 30.


RIDE - First test
File Edit Tools Navigate Macros Help
Test Suites
First test
TESTCASE0
External Resources
Files
Local Disk (C:)
Edit Text Edit Run
Execution Profile: robot
Report Log Autosave Pause on failure Show message log
Start Debug Stop Pause Continue Next Step over
Arguments:
Only run tests with these tags
Skip tests with these tags
elapsed time: 0:00:00 pass: 0 fail: 1
Saved all files

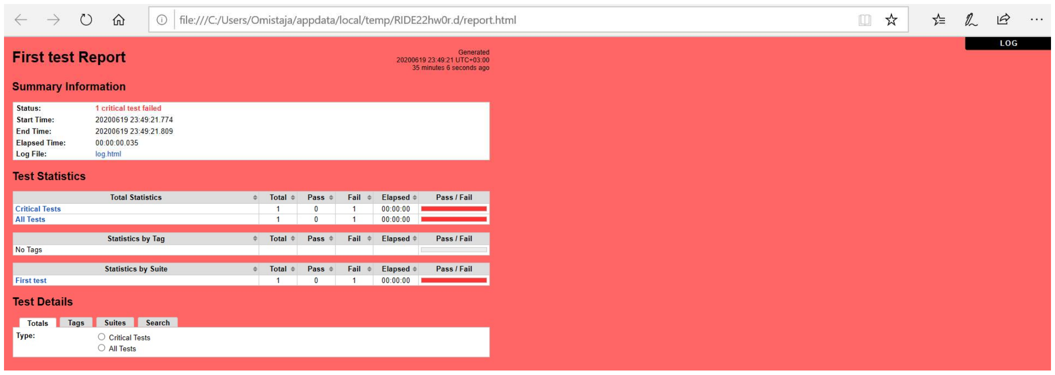
```
    BuiltIn.Log
------------------------------------------------------------------------------
First test
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
==============================================================================
Output:  c:\users\omistaja\appdata\local\temp\RIDE22hw0r.d\output.xml
Log:     c:\users\omistaja\appdata\local\temp\RIDE22hw0r.d\log.html
Report:  c:\users\omistaja\appdata\local\temp\RIDE22hw0r.d\report.html

test finished 20200619 23:49:22
```

```
Starting test: First test.TESTCASE0
20200619 23:49:21.804 :  INFO : Welcome to our first test
20200619 23:49:21.805 :  INFO : In robot framework
20200619 23:49:21.807 :  FAIL :
No keyword with name 'lo' found. Did you mean:
    BuiltIn.Log
Ending test:   First test.TESTCASE0
```

Figure 30. The failure of the test case.

Here the test case has failed. See the highlighted error in the screenshot above that it tells about the test case.

Now we will see the report and log output; when the test suit fails, the color is changed to red, as shown in figures 31 and 32. (Robot Framework, July 2020).


file:///C:/Users/Omistaja/appdata/local/temp/RIDE22hw0r.d/report.html

LOG

First test Report

Generated
20200619 23:49:21 UTC+03:00
35 minutes 6 seconds ago

Summary Information

Status:	1 critical test failed
Start Time:	20200619 23:49:21.774
End Time:	20200619 23:49:21.809
Elapsed Time:	00:00:00.035
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:00:00	🟥
All Tests	1	0	1	00:00:00	🟥

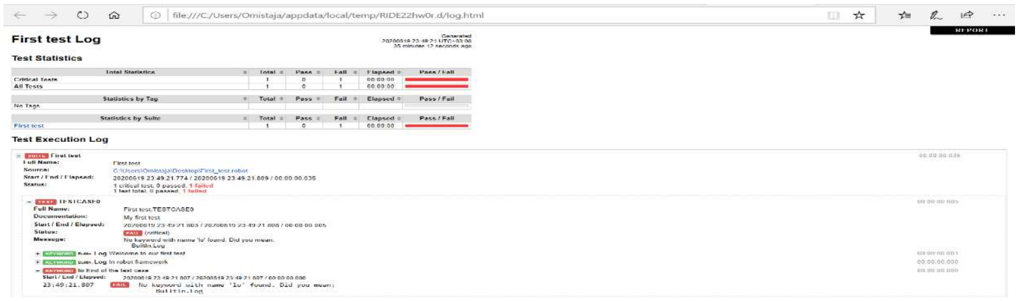
Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
First test	1	0	1	00:00:00	🟥

Test Details

Type: ○ Critical Tests ○ All Tests

Figure 31. The report file in a red color because of the failure.


file:///C:/Users/Omistaja/appdata/local/temp/RIDE22hw0r.d/log.html

REPORT

First test Log

Generated
20200619 23:49:21 UTC+03:00
35 minutes 12 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:00:00	🟥
All Tests	1	0	1	00:00:00	🟥

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
First test	1	0	1	00:00:00	🟥

Test Execution Log

```
- SUITE First test                                                        00:00:00.035
  Full Name:            First test
  Source:               C:\Users\Omistaja\Desktop\First_test.robot
  Start / End / Elapsed: 20200619 23:49:21.774 / 20200619 23:49:21.809 / 00:00:00.035
  Status:               1 critical test, 0 passed, 1 failed
                        1 test total, 0 passed, 1 failed
  - TEST TESTCASE0                                                        00:00:00.005
    Full Name:            First test.TESTCASE0
    Documentation:        My first test
    Start / End / Elapsed: 20200619 23:49:21.803 / 20200619 23:49:21.808 / 00:00:00.005
    Status:               FAIL (critical)
    Message:              No keyword with name 'lo' found. Did you mean:
                            BuiltIn.Log
    + KEYWORD BuiltIn.Log Welcome to our first test                       00:00:00.001
    + KEYWORD BuiltIn.Log In robot framework                              00:00:00.000
    - KEYWORD lo End of the test case                                     00:00:00.000
      Start / End / Elapsed: 20200619 23:49:21.807 / 20200619 23:49:21.807 / 00:00:00.000
      23:49:21.807  FAIL  No keyword with name 'lo' found. Did you mean:
                            BuiltIn.Log
```

Figure 32. The log file containing a red color because of the failure case.

It was a wonderful day; I could achieve my goal where I explained the features besides the first test case. The quantity of information that I gained during the day was great.

2.6.3 Sunday 12 July 2020: Working with Browsers Using Selenium

Selenium Library

Through my study about test automation, I found that most developers using the selenium library, so I found that it is important to recognize what this library is, which is today's goal.

It is a library specialized in testing webpages, but this use does not stop at the goal of the test only but extends it to include simulations of using the web, automation, and the operations carried out on the pages of websites and web systems in addition to scraping the web and collecting data.

Selenium works by running the browser and loading the site automatically and then performing various and multiple operations such as filling data in specific fields, reading the content of the site pages, implementing mouse and keyboard clicks on the page components, implementing JavaScript codes, navigating between pages and many other things.

Selenium does not have its own browser, and in order to work, a Firefox or Chrome browser is required, and if you want your program to run in the background, you can use the PhantomJS browser (called a headless browser). As it loads the site into memory and enables you to perform all operations available on regular browsers but without showing any graphics to the user.

The importance of Selenium

Selenium's importance is evident in several aspects. By using it in web testing and examining. You will most likely write a little code to test a large system according to the concept of blackbox automating testing or BAT. In contrast to other forms of testing, in which you may need to write a large software code to examine the properties of small in the system. This means we have reduced the time and effort needed to conduct blackbox testing.

Selenium offers an integrated framework for examining and testing web systems, and this facilitates the process of building applications that test other applications. Thus you will not repeat building many test codes, and you will not fall into the problem of maintaining and following up the test codes, all you have to do is build the test application in a correct way.

Selenium's importance also stems from the fact that it simulates the real user behavior of the site. Some automation frameworks inject the browser with software codes to simulate the user's operations, such as pressing a specific button. Also, Selenium supports a wide range of the most popular browsers on different platforms such as Linux and Windows.

We can summarize the importance of Selenium as follows:

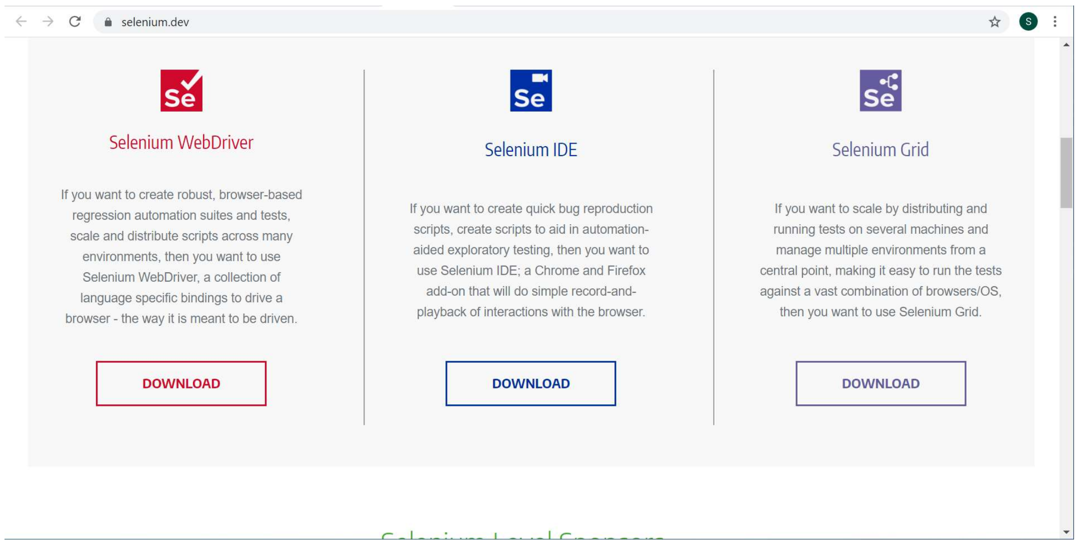
- Saving time and effort
- Ease of use
- Compatibility with different browsers
- Working in many operating environments
- They are used for checking, automating, and simulating user behavior.

Generally, Today I was able to obtain abundant and sufficient information about this wonderful library, and I think I am ready to go ahead with this library and study its instructions in detail.

2.6.4 Monday 13 July 2020: Test Case Using Chrome Browser

Today I have been thinking to take another practical example about a test case using a browser, but I also found I have to talk about installing drivers of a browser. So, my objectives for today will be to install a driver for a browser in addition to a practical example.

We need to install the drivers for chrome to work with Selenium. We can download them from Selenium official website https://www.seleniumhq.org. See figure 33.



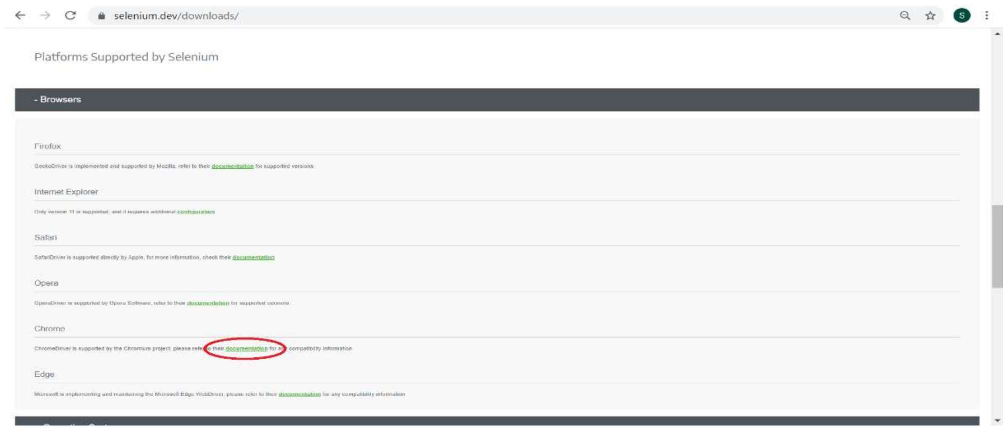
If you want to create robust, browser-based regression automation suites and tests, scale and distribute scripts across many environments, then you want to use Selenium WebDriver, a collection of language specific bindings to drive a browser - the way it is meant to be driven.

If you want to create quick bug reproduction scripts, create scripts to aid in automation-aided exploratory testing, then you want to use Selenium IDE; a Chrome and Firefox add-on that will do simple record-and-playback of interactions with the browser.

If you want to scale by distributing and running tests on several machines and manage multiple environments from a central point, making it easy to run the tests against a vast combination of browsers/OS, then you want to use Selenium Grid.

Figure 33. Selenium official website.

In the part of Selenium WebDriver, click download and got to the Platforms Supported by Selenium on the Browsers select your browser, press on documentation, and download the latest version. Here I will use chrome browser, as shown in figure 34.



selenium.dev/downloads/

Platforms Supported by Selenium

- Browsers

Firefox

GeckoDriver is implemented and supported by Mozilla, refer to their documentation for supported versions.

Internet Explorer

Only version 11 is supported, and it requires additional configuration

Safari

SafariDriver is supported directly by Apple, for more information, check their documentation

Opera

OperaDriver is supported by Opera Software, refer to their documentation for supported versions.

Chrome

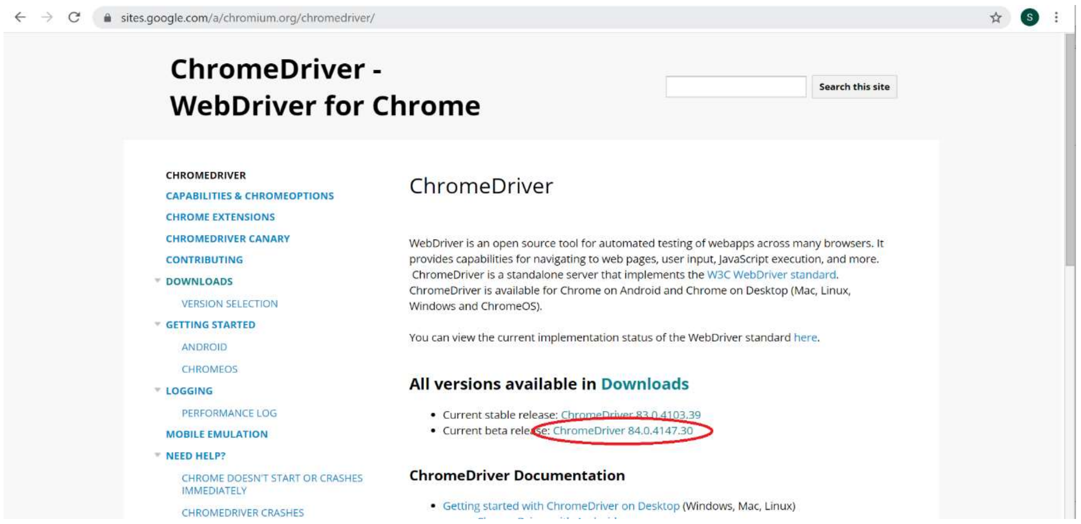
ChromeDriver is supported by the Chromium project, please refer to their documentation for any compatibility information

Edge

Microsoft is implementing and maintaining the Microsoft Edge WebDriver, please refer to their documentation for any compatibility information

Figure 34. How to download chrome supported by Selenium.

Select the latest version. It will show the downloads as per the operating system. See figures 35 and 36.



sites.google.com/a/chromium.org/chromedriver/

ChromeDriver -
WebDriver for Chrome

ChromeDriver

WebDriver is an open source tool for automated testing of webapps across many browsers. It provides capabilities for navigating to web pages, user input, JavaScript execution, and more. ChromeDriver is a standalone server that implements the W3C WebDriver standard. ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows and ChromeOS).

You can view the current implementation status of the WebDriver standard here.

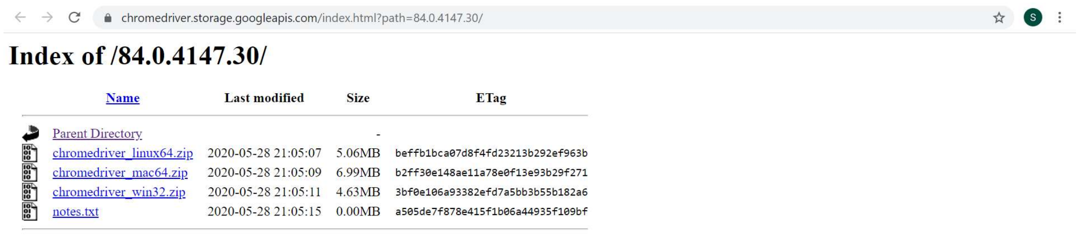
All versions available in Downloads

- Current stable release: ChromeDriver 83.0.4103.39
- Current beta release: ChromeDriver 84.0.4147.30

ChromeDriver Documentation

- Getting started with ChromeDriver on Desktop (Windows, Mac, Linux)

Figure 35. The downloads as per the operating system.



chromedriver.storage.googleapis.com/index.html?path=84.0.4147.30/

Index of /84.0.4147.30/

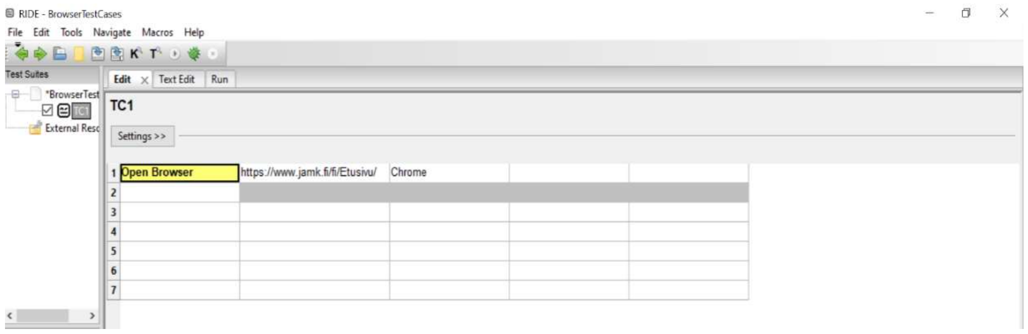
Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2020-05-28 21:05:07	5.06MB	beffb1bca07d8f4fd23213b292ef963b
chromedriver_mac64.zip	2020-05-28 21:05:09	6.99MB	b2ff30e148ae11a78e0f13e93b29f271
chromedriver_win32.zip	2020-05-28 21:05:11	4.63MB	3bf0e106a93382efd7a5bb3b55b182a6
notes.txt	2020-05-28 21:05:15	0.00MB	a505de7f878e415f1b06a44935f109bf

Figure 36. The downloadable files.

Download the version according to your operating system from the above list. It will download as a zip file. When the file is downloaded completely, unzip it and copy the .exe driver file to the python folder (C:\Python27).

Now we have installed the driver for chrome. We can start with writing test cases that will open and close the browser.

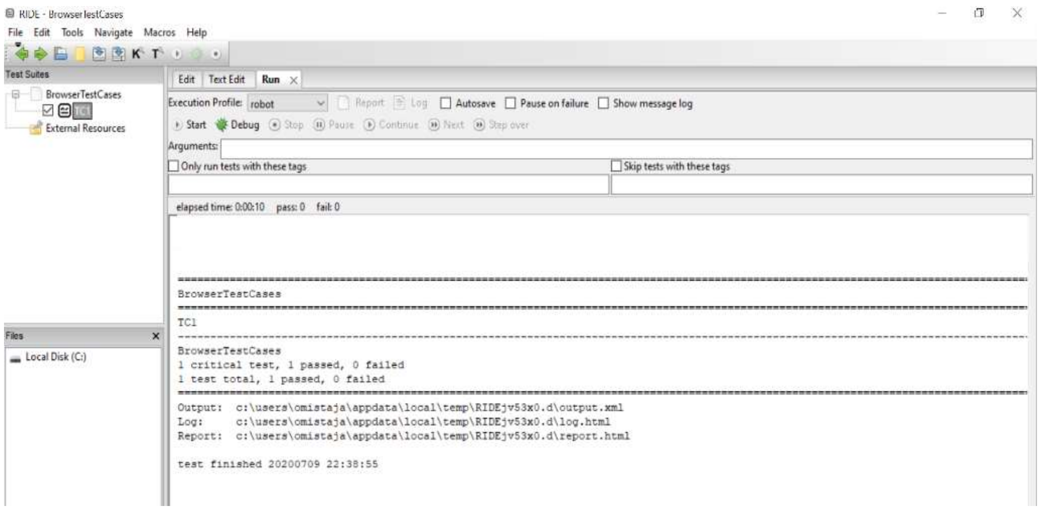
First open Ride, and then in the test case, we will open the site "https://www.jamk.fi/fi/Etusivu/" in chrome, and the test case details will be, as shown in figure 37. (Robot — Working With Browsers Using Selenium Library, July 2020).



1	Open Browser	https://www.jamk.fi/fi/Etusivu/	Chrome		
2					
3					
4					
5					
6					
7					

Figure 37. The test case details.

Now we will run the test case and see the result, as shown in figures 38 and 39.



Execution Profile: robot ☐ Autosave ☐ Pause on failure ☐ Show message log

Arguments:

☐ Only run tests with these tags ☐ Skip tests with these tags

elapsed time: 0:00:10 pass: 0 fail: 0

```
==============================================================================
BrowserTestCases
==============================================================================
TC1
------------------------------------------------------------------------------
BrowserTestCases
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\output.xml
Log:     c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\log.html
Report:  c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\report.html

test finished 20200709 22:38:55
```

Figure 38. The result of running the test case.

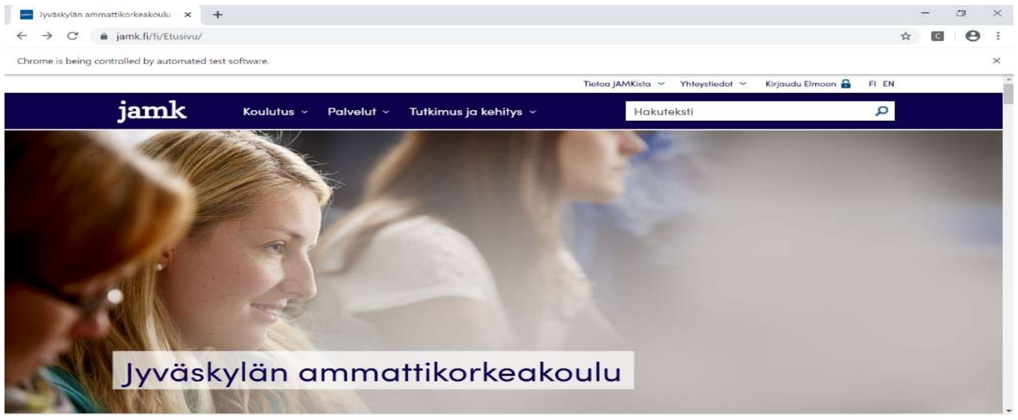
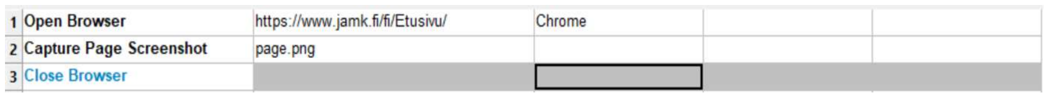


Figure 39. The website we tested.

When the test case pass, we will see the chrome browser will open with the tested website. Now we will add more tests, as shown in figure 40.



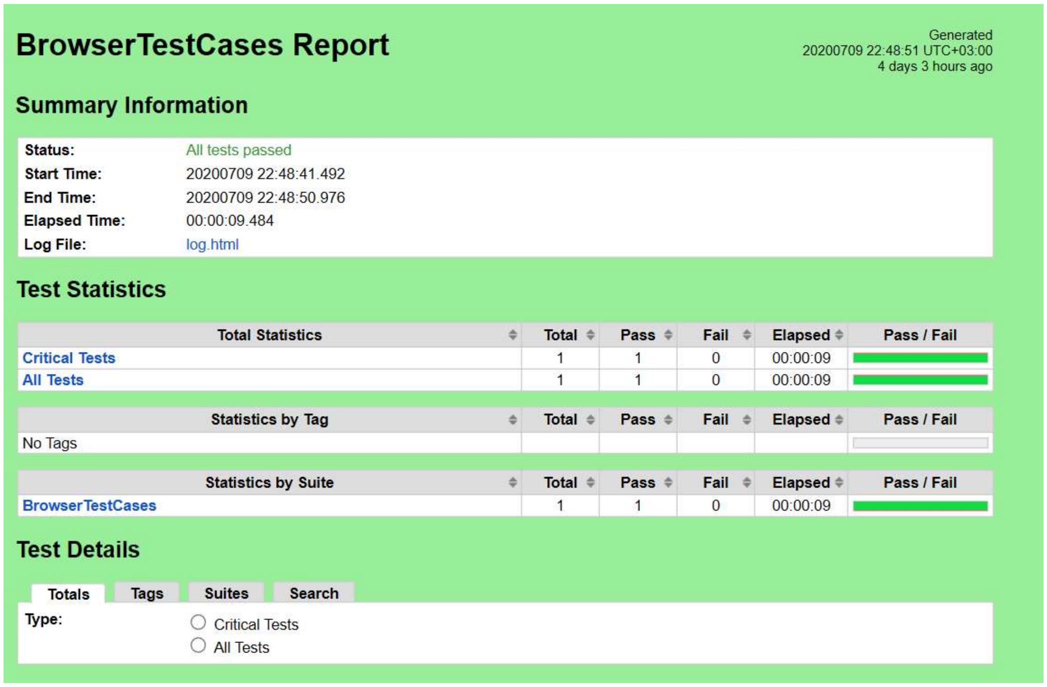
1	Open Browser	https://www.jamk.fi/fi/Etusivu/	Chrome		
2	Capture Page Screenshot	page.png			
3	Close Browser				

Figure 40. Adding more tests cases.

- Open Browser: will open URL “https://www.jamk.fi/fi/Etusivu/” in Chrome browser.
- Capture Page Screenshot: will take a screenshot for the website and the image’s name is page.png.
- The last is close browser.

After running the test cases above we will get the details of report and log, as shown in figures 41 and 42.

Report:



BrowserTestCases Report

Generated
20200709 22:48:51 UTC+03:00
4 days 3 hours ago

Summary Information

Status:	All tests passed
Start Time:	20200709 22:48:41.492
End Time:	20200709 22:48:50.976
Elapsed Time:	00:00:09.484
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:09	🟩
All Tests	1	1	0	00:00:09	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
BrowserTestCases	1	1	0	00:00:09	🟩

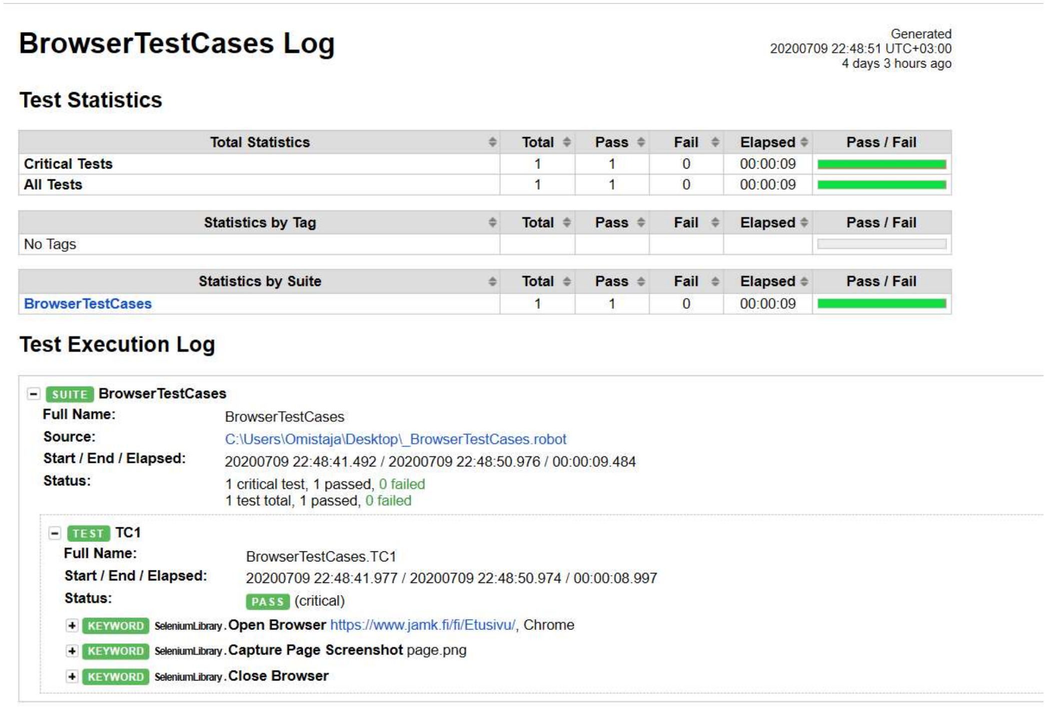
Test Details

Totals Tags Suites Search

Type: ○ Critical Tests ○ All Tests

Figure 41. The details of report .

Log:



BrowserTestCases Log

Generated
20200709 22:48:51 UTC+03:00
4 days 3 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:09	🟩
All Tests	1	1	0	00:00:09	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

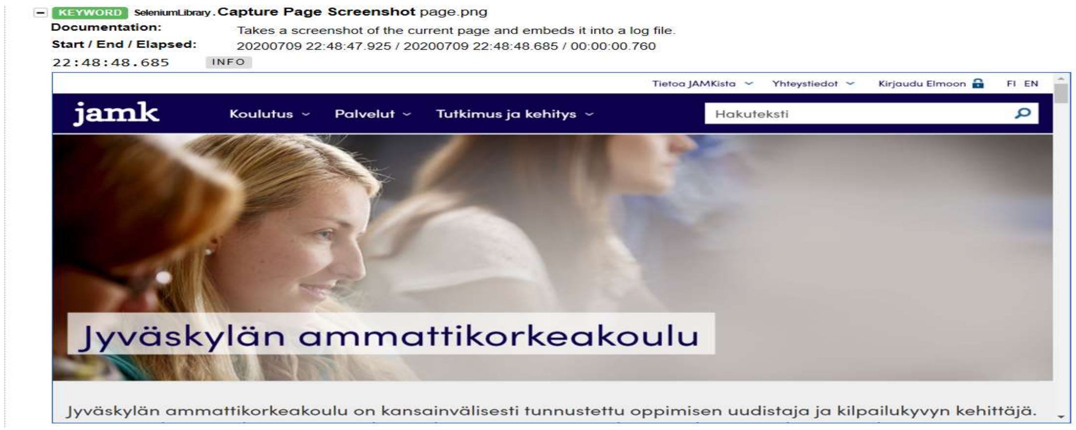
Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
BrowserTestCases	1	1	0	00:00:09	🟩

Test Execution Log

```
− SUITE BrowserTestCases
   Full Name:              BrowserTestCases
   Source:                 C:\Users\Omistaja\Desktop\_BrowserTestCases.robot
   Start / End / Elapsed:  20200709 22:48:41.492 / 20200709 22:48:50.976 / 00:00:09.484
   Status:                 1 critical test, 1 passed, 0 failed
                           1 test total, 1 passed, 0 failed

   − TEST TC1
      Full Name:              BrowserTestCases.TC1
      Start / End / Elapsed:  20200709 22:48:41.977 / 20200709 22:48:50.974 / 00:00:08.997
      Status:                 PASS (critical)
      + KEYWORD SeleniumLibrary.Open Browser https://www.jamk.fi/fi/Etusivu/, Chrome
      + KEYWORD SeleniumLibrary.Capture Page Screenshot page.png
      + KEYWORD SeleniumLibrary.Close Browser
```

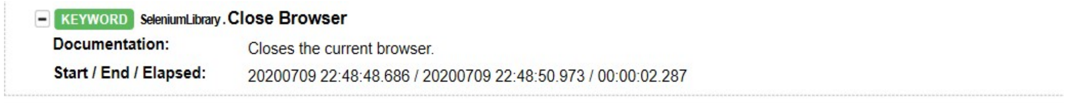
Figure 42. The details of the log.

Details of log:

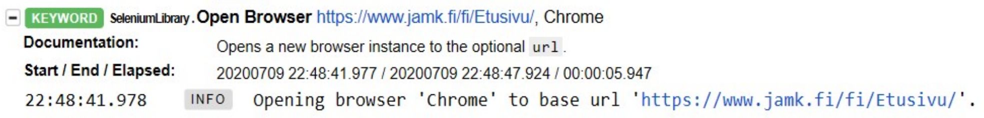


```
KEYWORD SeleniumLibrary.Capture Page Screenshot page.png
Documentation:        Takes a screenshot of the current page and embeds it into a log file.
Start / End / Elapsed: 20200709 22:48:47.925 / 20200709 22:48:48.685 / 00:00:00.760
22:48:48.685  INFO
```

Tietoa JAMKista
Yhteystiedot
Kirjaudu Elmoon
FI EN
jamk
Koulutus
Palvelut
Tutkimus ja kehitys
Hakuteksti
Jyväskylän ammattikorkeakoulu
Jyväskylän ammattikorkeakoulu on kansainvälisesti tunnustettu oppimisen uudistaja ja kilpailukyvyn kehittäjä.



```
KEYWORD SeleniumLibrary.Close Browser
Documentation:        Closes the current browser.
Start / End / Elapsed: 20200709 22:48:48.686 / 20200709 22:48:50.973 / 00:00:02.287
```

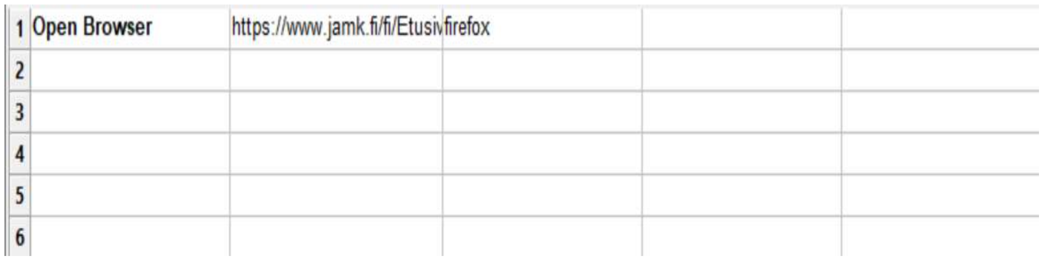


```
KEYWORD SeleniumLibrary.Open Browser https://www.jamk.fi/fi/Etusivu/, Chrome
Documentation:        Opens a new browser instance to the optional url.
Start / End / Elapsed: 20200709 22:48:41.977 / 20200709 22:48:47.924 / 00:00:05.947
22:48:41.978  INFO  Opening browser 'Chrome' to base url 'https://www.jamk.fi/fi/Etusivu/'.
```

By the end of the day, I can say that my knowledge becomes better, and I understood the idea behind the Robot Framework because the quantity of the information I gained was really wonderful and helpful.

2.6.5 Tuesday 14 July 2020: Test Case Using Firefox Browser

Today is a continuation of yesterday because I found it important to touch upon the use of the test using Firefox to cover the topic of working with browsers.
We have to install the Firefox driver and save it in the python folder. Firefox test case and the result as shown in figures 43 and 44.



1	Open Browser	https://www.jamk.fi/fi/Etusiv	firefox		
2					
3					
4					
5					
6					

Figure 43. The test cases using Firefox browser.

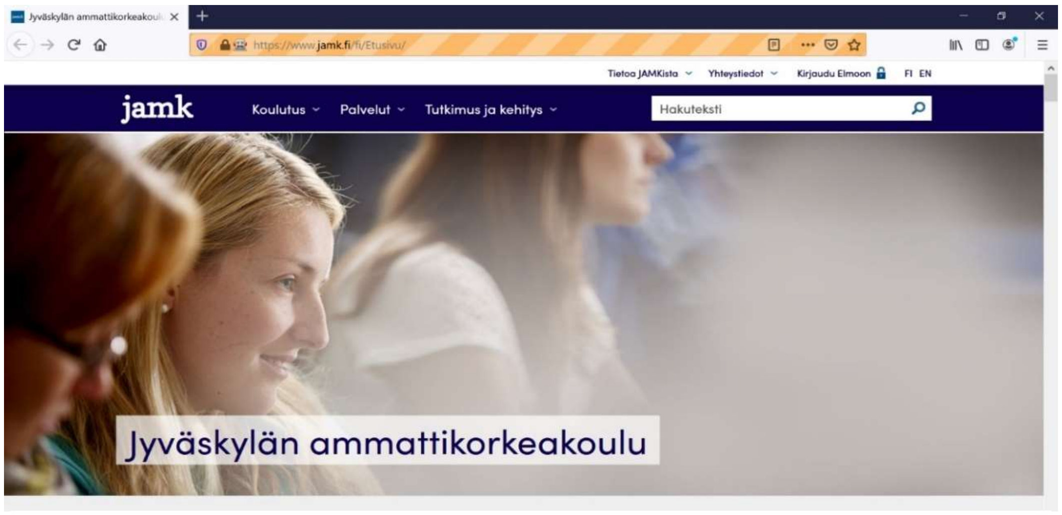


Figure 44. The result using Firefox browser.

2.6.6 Wednesday 15 July 2020: Working with Setup and Teardown

Today's objective will be explaining the important rest features of the Ride editor with an example to finish this chapter and move to the last part of this thesis.

Setup

This is a set of keywords to be implemented before the start of test suite or a test case implementation.

Teardown

This is a set of keywords to be implemented after the start of test suite or a test case implementation.

We will work on a project setup and use setup and teardown. The opening and closing of the browser are the most common steps in test cases.

Now, we will add the keyword open browser in the case of setup and close browser in the case of the teardown.

Now, open Ride and create a new project, then create a test case. Also, we need to import Selenium Library to use the keywords concerning the browser and interact with the pages. See figure 45. (Robot Framework, Working With Setup And Teardown, July 2020).

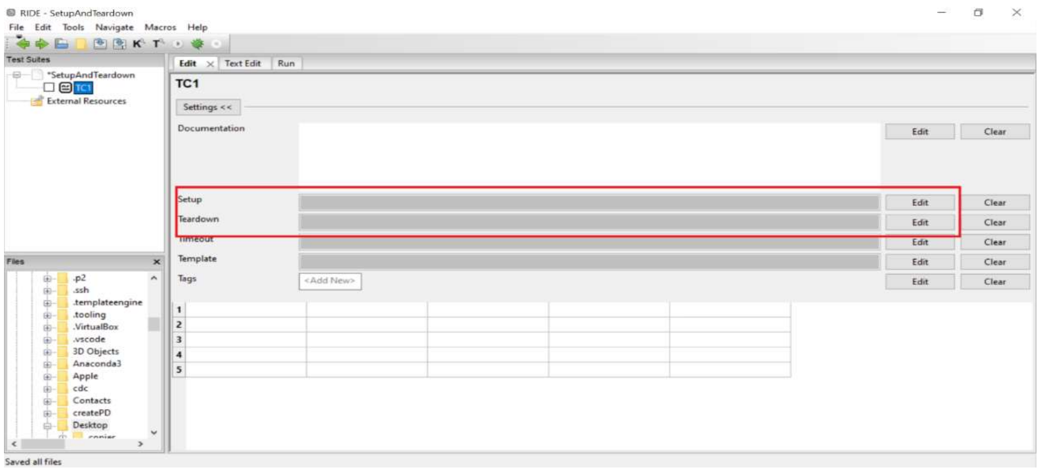
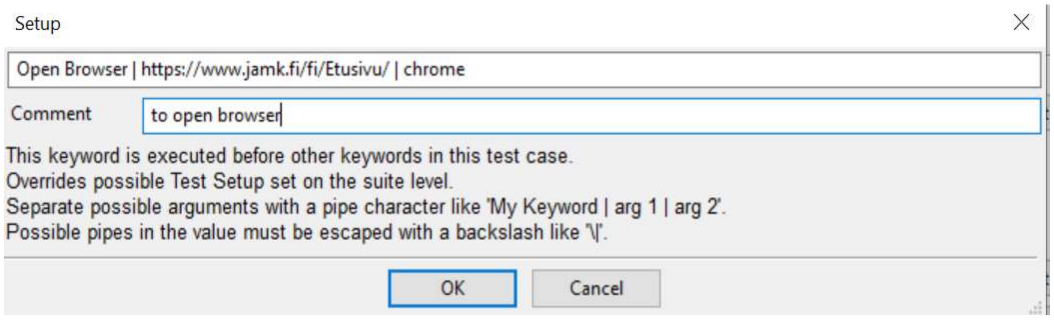


Figure 45. The Setup and the Teardown.

In the above screenshot, we have options in the settings section, and here we will use the Setup and the Teardown. Click Edit to enter the keyword, and arguments must be separated with the pipe character (|), as shown in figure 46. (Robot Framework, Working With Setup And Teardown,July 2020).


Setup

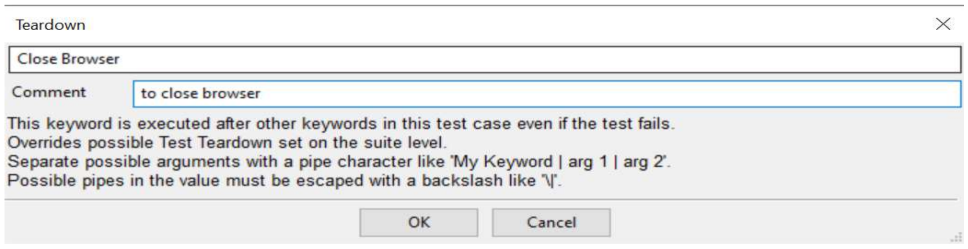
Open Browser | https://www.jamk.fi/fi/Etusivu/ | chrome

Comment: to open browser

This keyword is executed before other keywords in this test case.
Overrides possible Test Setup set on the suite level.
Separate possible arguments with a pipe character like 'My Keyword | arg 1 | arg 2'.
Possible pipes in the value must be escaped with a backslash like '\|'.

Figure 46. Arguments separated with the pipe character.

In the teardown case, we will enter the close browser keyword, as shown in figure 47.


Teardown

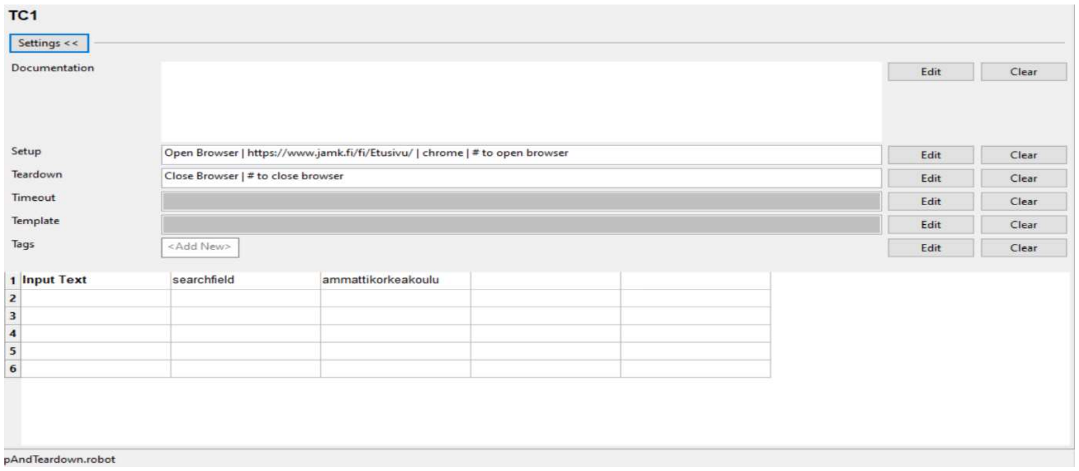
Close Browser

Comment: to close browser

This keyword is executed after other keywords in this test case even if the test fails.
Overrides possible Test Teardown set on the suite level.
Separate possible arguments with a pipe character like 'My Keyword | arg 1 | arg 2'.
Possible pipes in the value must be escaped with a backslash like '\|'.

Figure 47. Teardown case.

Then, we will write the keywords to the test case, as shown in figure 48.


TC1

Settings <<

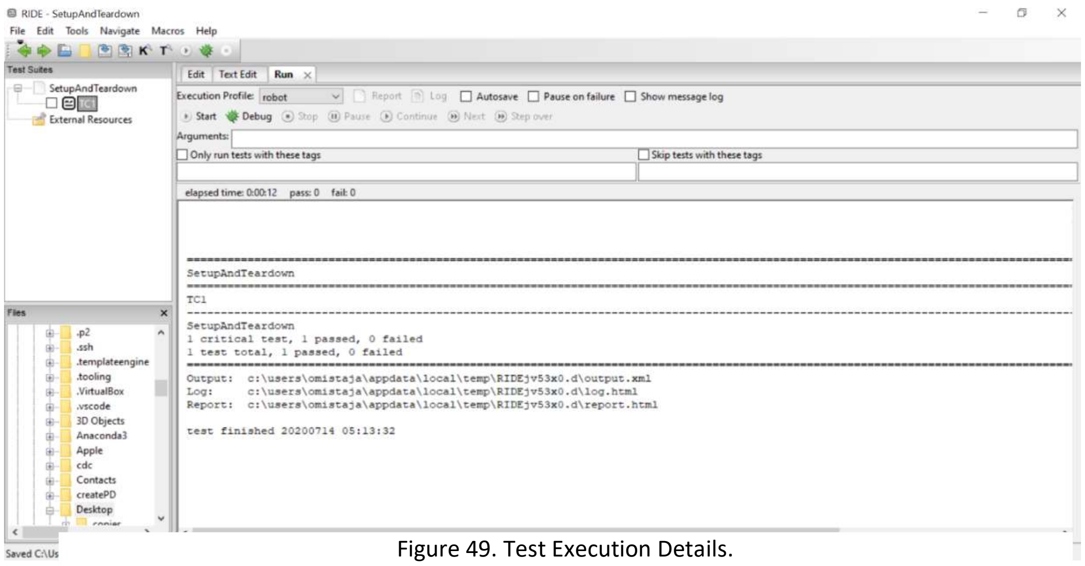
Documentation	
Setup	Open Browser | https://www.jamk.fi/fi/Etusivu/ | chrome | # to open browser
Teardown	Close Browser | # to close browser
Timeout	
Template	
Tags	<Add New>

1	Input Text	searchfield	ammattikorkeakoulu		
2					
3					
4					
5					
6					

pAndTeardown.robot

Figure 48. Writing the keywords to the test suite.

We have Input Text in the test case. The opening and closing of the browser are coming from Setup and Teardown Settings. Below are Test Execution Details. See figure 49. (Robot Framework, Working With Setup And Teardown, July 2020).



Execution Profile: robot

Arguments:

elapsed time: 0:00:12 pass: 0 fail: 0

```
==============================================================================
SetupAndTeardown
==============================================================================
TC1
------------------------------------------------------------------------------
SetupAndTeardown
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\output.xml
Log:     c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\log.html
Report:  c:\users\omistaja\appdata\local\temp\RIDEjv53x0.d\report.html

test finished 20200714 05:13:32
```

Figure 49. Test Execution Details.

By the end, I can say that I have completed all the important features required, which I have applied with example. The information that I got was just to complete the features of this editor and finish this chapter completely.

2.6.7 Weekly Analysis

This week I have completed the Robot Framework chapter, which I consider as the core of the thesis. I have focused on the recognition of the Selenium library and working with browsers.
I touched on practical examples and explained some of the Ride editor features. I noticed how Robot works and learned how to test an application using Chrome and Firefox browsers. According to what I got this week, I think I need to address more tests in the future to be professional.

2.7 Week 7

2.7.1 Thursday 17 July 2020: GitLab

This will be the last chapter related to this thesis. Generally, GitLab is not new for me. I have used it before, and I have its basics. But this time, I will use CI/CD pipeline. Before that, I have to explain the general idea behind it and the basics of GitHub and GitLab, which they will be my objectives for today.
GitLab is a web DevOps lifecycle tool that gives a Git-repository manager providing wiki, issue-tracking, and continuous integration, which called CI/continuous deployment, which called CD pipeline, using an open-source license, developed by GitLab Inc. Ukrainians Dmitriy Zaporozhets and Valery Sizov programmed the software.
The code was written in Ruby, with some parts later rewritten in Go, initially as a source code management solution to cooperate with his team on software development. It later developed to an integrated solution containing the software development lifecycle and then to the fully DevOps lifecycle. The current machinery stack includes Go, Ruby on Rails, and Vue.js.
It follows an open-source development model where the main functionality is released under an open-source (MIT) license while the additional functionality is under a proprietary license.

2.7.2 The Basics of GitHub and GitLab

GitHub is a web-based repository holding platform with 26 million users (March 2017). Originally, GitHub started in 2008 and was founded by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett.

GitHub projects can become public, and every shared code is freely open to everybody. You can create private projects as well, but only three contributors are allowed on the free plan.

Generally, 26 million people and more than 1.5 million organizations created 67 million repositories on GitHub in 2017, see figure 50. (GitLab vs GitHub: Key differences & similarities, July 2020).

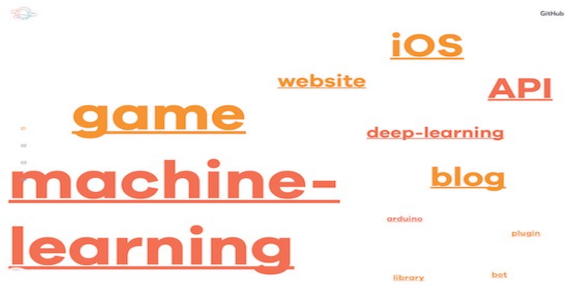


Figure 50. The most used repository tags on GitHub.com.

source: https://usersnap.com/blog/gitlab-github/

Like GitHub, GitLab is a repository manager that allows teams to cooperate on code. Written in Ruby and Go. GitLab offers some similar features for issue tracking and project management as GitHub.

GitLab is founded in 2011 by Dmitriy Zaporozhets and Valery Sizov. The employs of GitLab quite 500 people and has quite 1,400 open source contributors. Consistent with Wikipedia, GitLab has more than 100,000 users (March 2017) and is used by enterprises like IBM, Sony, and NASA. (GitLab vs GitHub: Key differences & similarities, July 2020)

Finally, I spent a lot of time reading about GitLab and GitHub and managed to learn a lot about them in addition to my previous knowledge. I can say that I got a good amount of important information.

2.7.3 Friday 18 July 2020: GitHub vs GitLab

Yesterday was a totally theoretical day, today's goal will also be theoretical, which is the difference between GitLab and GitHub. I will address these differences from several aspects, the most important of which are:

Authentication Levels: With GitLab, you can set and alter people's permissions consistent with their role. In GitHub, you will determine if someone gets a read or write access to a repository.

With GitLab, you can give access to the issue tracker (for example) without giving permission to the source code. This is clearly great for larger teams and enterprises with role-based contributors.

Built-in CI / CD & going Beyond CD: One of the main differences between GitLab and GitHub is the built-in CI/CD of GitLab. CI is a great time saver for a lot of development teams and a great way of QA.

GitLab gives its very own CI for free. So, you do not need to use an external CI service. And if you are using an external CI, you can integrate with Jenkins, Buddy, Codeship, and others.

As GitLab mentioned with its latest release is clearly deals with the DevOps market besides offering an operations dashboard that allows you to understand the dependencies of your development and DevOps efforts, see figure 51. (GitLab vs GitHub: Key differences & similarities, July 2020).

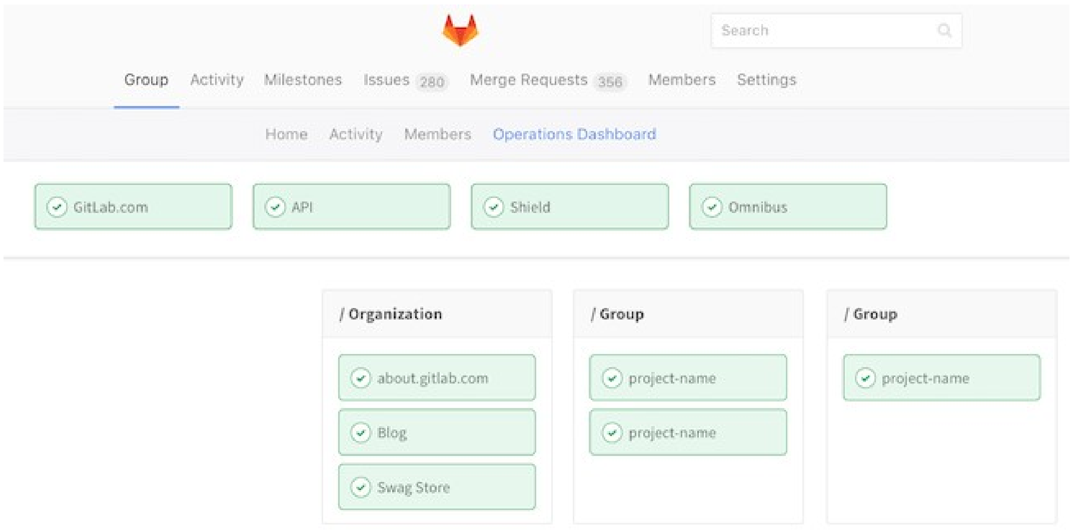


Figure 51. Operations Dashboard

source: https://usersnap.com/blog/gitlab-github/

Also, GitLab is dealing with the topic of Auto CI and how to automatically run CI/CD without a person setting it up.

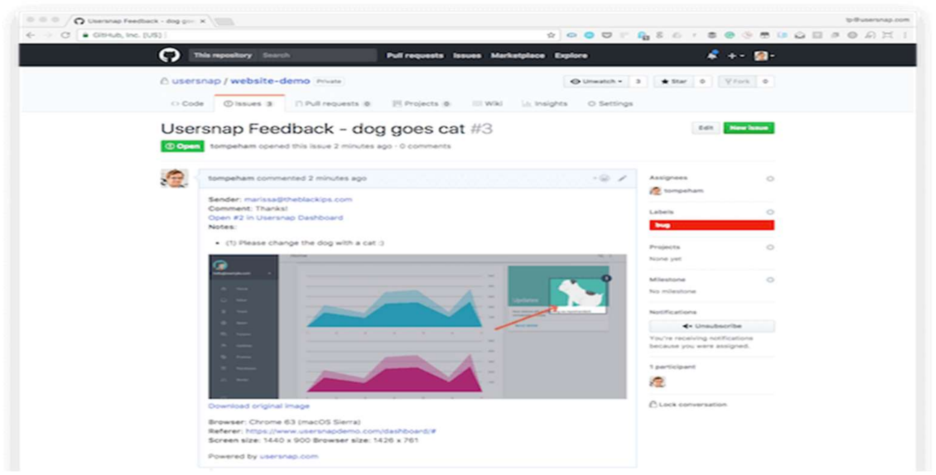
"But, really, every project should be running some kind of CI. So, why don't we just detect when you've pushed up a project; we'll just build it and we'll go and test it,

because we know how to do testing".
Mark Pundsack, source: gitlab.com

Issue Tracking: GitLab and GitHub provide a simple issue tracker that permits you to change status and assignee for several issues all together.

Both are wonderful issue trackers, especially when associated with a visual bug tracker like Usersnap. While your developers still enjoy the good issue tracking interface of GitLab & GitHub, your testers and users can simply report bugs through the Usersnap tool.

Reports of Bug and user feedback can be sent automatically to GitLab or GitHub. Or you can pre-filter them inside Usersnap and manually send them to your development project, see figures 52 and 53.


Usersnap Feedback - dog goes cat #3

tompeham opened this issue 2 minutes ago · 0 comments

tompeham commented 2 minutes ago

Sender: marissa@theblackips.com
Comment: Thanks!
Open #2 in Usersnap Dashboard
Notes:

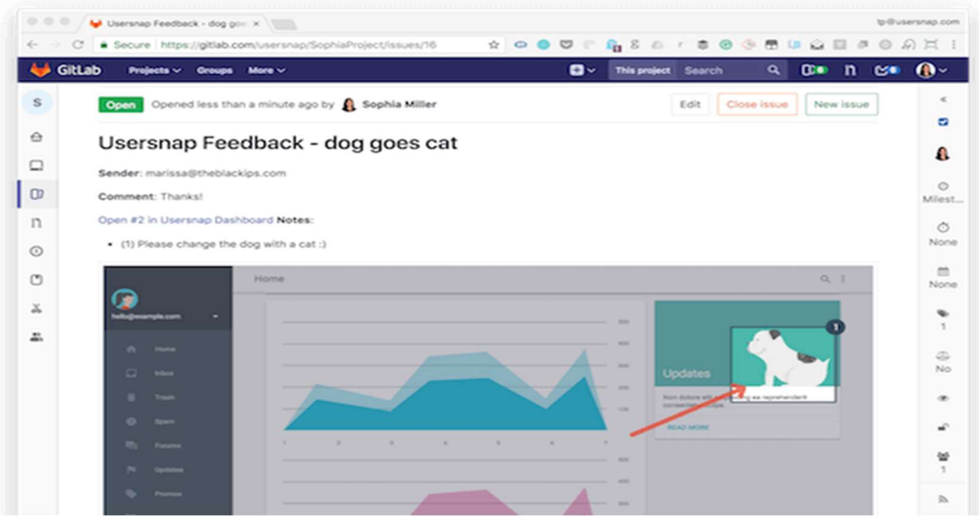
- (1) Please change the dog with a cat :)

Download original image

Browser: Chrome 63 (macOS Sierra)
Referer: https://www.usersnapdemo.com/dashboard/#
Screen size: 1440 x 900 Browser size: 1426 x 761

Powered by usersnap.com

Figure 52. Incoming bug reports displayed in your GitHub repository.
source: https://usersnap.com/blog/gitlab-github/


Open Opened less than a minute ago by Sophia Miller

Usersnap Feedback - dog goes cat

Sender: marissa@theblackips.com

Comment: Thanks!

Open #2 in Usersnap Dashboard Notes:

- (1) Please change the dog with a cat :)

Figure 53. Incoming bug reports displayed in your GitHub repository.

source: https://usersnap.com/blog/gitlab-github/

Import & Export: When deciding about moving to GitLab or GitHub, you ought to consider the setup costs; and resources needed for getting started. Therein regard, the subject of obtainable import and export features are pretty important.
GitLab presents detailed documentation on how to import your data from other suppliers like GitHub to GitLab, see figure 54. (GitLab vs GitHub: Key differences & similarities, July 2020)

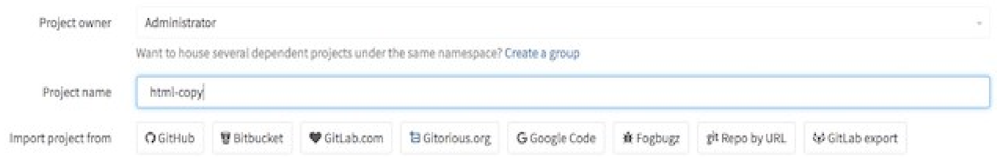

Project owner: Administrator
Want to house several dependent projects under the same namespace? Create a group
Project name: html-copy
Import project from: GitHub | Bitbucket | GitLab.com | Gitorious.org | Google Code | Fogbugz | Repo by URL | GitLab export

Figure 54. Import projects from 3rd parties.

source: https://usersnap.com/blog/gitlab-github/

GitHub, on the opposite hand, does not offer like detailed documentation for the more common git repositories. However, GitHub presents to use GitHub Importer if you've got your source code in Subversion, Mercurial, TFS, and others.
Also, when it involves exporting data, GitLab looks to do a good job, offering you the power to export your projects, including the subsequent data:

- Wiki and project repositories
- Project uploads
- The configuration including webhooks and services
- Issues with comments, merge requests with diffs and comments, labels, milestones, snippets, and other project entities.

On the other hand, GitHub looks to be more restrictive when it involves export features of existing GitHub repositories. (GitLab vs GitHub: Key differences & similarities, July 2020)
Integrations: Both GitLab and GitHub offer a good range of 3rd party integrations. Integrating your version system with other application enhance your workflows and may boost productivity for your developers and your non-developers.
In order to see out if your favorite apps are compatible with GitLab and GitHub, I like to recommend reading the documentation of GitLab and GitHub.

Further to the available integration partners, GitHub released GitHub marketplace in May 2017, presenting you with chose tools and applications.

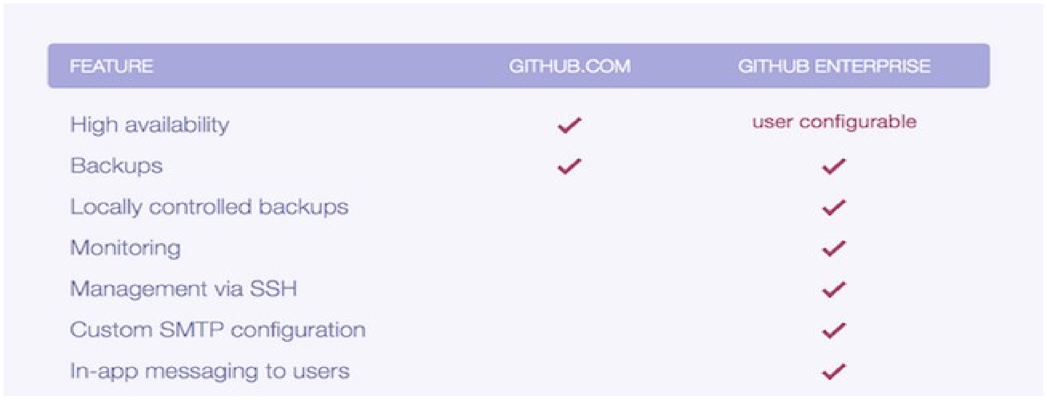
GitLab took an identical path and offered multiple integrations for development and DevOps teams. (GitLab vs GitHub: Key differences & similarities, July 2020)

The GitHub community: GitHub situated itself among its community of developers. And its popularity is especially driven by the highly active GitHub community of many developers. You will discuss problems and perhaps learn a couple of unofficial but awesome hacks there. On the opposite hand, GitLab undertook some wonderful activities, like holding community events and connecting open source contributors. If you are searching for the biggest community of developers, the chances are high that GitHub is the better place to be.

If you are trying to find the most important community of developers, the likelihood is that GitHub is that the better place to be. (GitLab vs GitHub: Key differences & similarities, July 2020)

GitLab Enterprise vs GitHub Enterprise: On an enterprise level, you ought to consider further factors when you want to make a decision of whether to use GitLab vs GitHub.

GitHub is very popular among developers, and over a previous couple of years, it gained popularity among larger development teams and organizations too. See figure 55. (GitLab vs GitHub: Key differences & similarities, July 2020)



FEATURE	GITHUB.COM	GITHUB ENTERPRISE
High availability	✓	user configurable
Backups	✓	✓
Locally controlled backups		✓
Monitoring		✓
Management via SSH		✓
Custom SMTP configuration		✓
In-app messaging to users		✓

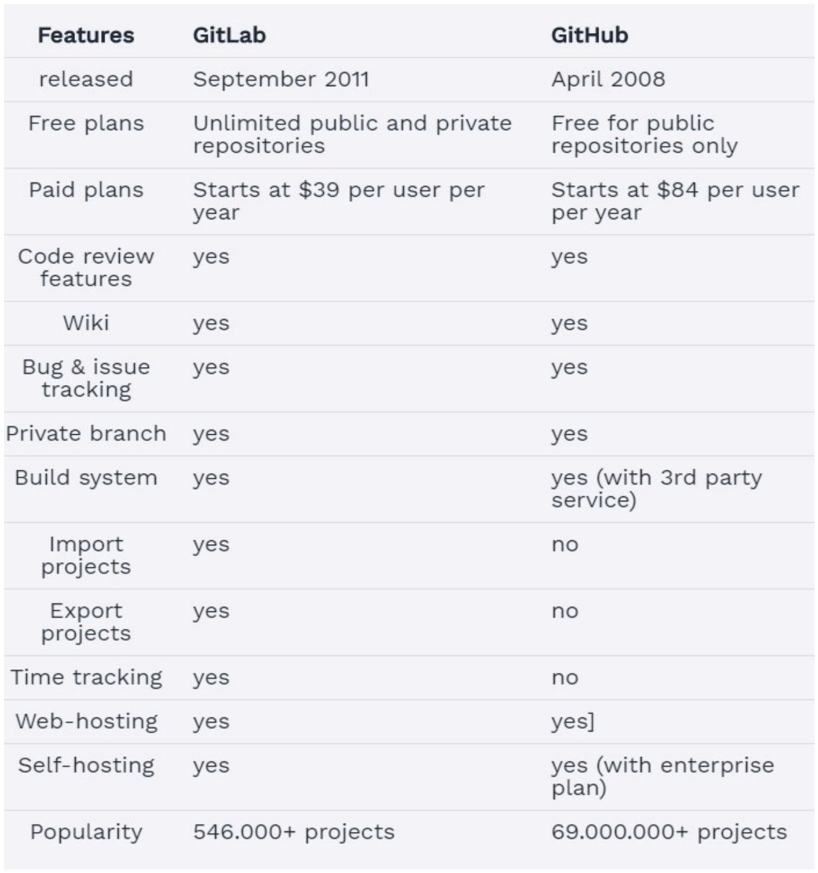
Figure 55. Custom enterprise features of GitHub

source: https://usersnap.com/blog/gitlab-github/

On the opposite hand, GitLab is very strong on enterprise features, too. With different enterprise plans available, GitLab is especially popular among larger development teams.

Below is how GitLab and GitHub compare pricing.

While the enterprise of GitHub plan starts at 2,500 USD per 10 users per year (= 250 USD per user), the enterprise of GitLab starter plan is 39 USD per user/per year. See figure 56. (GitLab vs GitHub: Key differences & similarities, July 2020)



Features	GitLab	GitHub
released	September 2011	April 2008
Free plans	Unlimited public and private repositories	Free for public repositories only
Paid plans	Starts at $39 per user per year	Starts at $84 per user per year
Code review features	yes	yes
Wiki	yes	yes
Bug & issue tracking	yes	yes
Private branch	yes	yes
Build system	yes	yes (with 3rd party service)
Import projects	yes	no
Export projects	yes	no
Time tracking	yes	no
Web-hosting	yes	yes]
Self-hosting	yes	yes (with enterprise plan)
Popularity	546.000+ projects	69.000.000+ projects

Figure 56. The enterprise of GitHub and GitLab starter plan

source: https://usersnap.com/blog/gitlab-github/

Wrapping it up: Undoubtedly, GitHub remains the most popular git repository with the most important number of users and projects. However, GitLab is doing a great job offering your development teams great tools for more useful workflows.

Today went well. I learned the differences between the most popular DevOps webs. Today's outcome was very impressive. (GitLab vs GitHub: Key differences & similarities, July 2020)

2.7.4 Wednesday 22 July 2020: Continuous integration and delivery

Today will be so short where I only dedicated it to understanding continuous integration and continuous delivery in general.

Continuous integration and continuous delivery enable DevOps teams to extend software development speed and deliver better quality code quicker. Continuous integration works to integrate code from your team during a shared repository, vastly improving your deployment pipeline. Developers share their new code during a Merge Request, which triggers a pipeline to create, test, and validate the new code before merging the changes in your repository. Continuous delivery deploys the code of CI-validated to your application.

All code is tested all over each stage, guaranteeing better quality builds and applications with less bugs. CI/CD pipelines can identify what happens when builds pass or fail these tests, which means that errors are identified much faster. As code runs all over each stage of the development process, it is continually validated compared to many other changes in the repository happening concurrently, which guarantees code integrity through the pipeline. Together, CI and CD speed up how quickly your team delivers results for your customers and stakeholders.

Continuous delivery is often utilized interchangeably with a continuous deployment release process, but there is a subtle difference between the two. Continuous deployment indicates that all validated code deploys to production automatically, whereas continuous delivery means that this code can be deployed. The flexibility for code to be deployed at any time is what distinguishes delivery from deployment, and practicing continuous deployment is possible when continuous delivery is already in place. (Continuous integration and delivery1, July 2020).

Although the day was short, I found many important sources which are talking about the importance of CI/CD. The information I gained today was enough to understand what CI/CD pipeline is.

2.7.5 Weekly Analysis

This week was totally theoretical. I read so many sources concerning GitLab and GitHub. The information I got was abundant. I addressed in detail what Gitlab is, which is clarified the idea behind it.

Then I touched on the differences between GitLab and GitHub from different aspects; the most important of them were authentication levels, Built-in CI / CD, issue tracking...etc.
Afterward, I addressed the concept of continuous integration and continuous delivery, which I consider the goal of this thesis.
Finally, this week was interesting. Despite I do not like, theoretical side, but I have seen some videos related to CI/CD pipeline to be more interesting and feel fun during the reading. My question now is: 'What are things that I need to know about CI/CD?' to be more familiar with this important aspect.

2.8 Week 8

2.8.1 Thursday 23 July 2020: Benefits of CI/CD

Today I will continue with the theoretical part, which is related to CI/CD pipeline. So, the objectives that I set for today are:

- Benefits of CI/CD
- The importance of GitLab CI/CD
- The definition of Pipeline.

CI/CD automates workflows and lessens error rates in a production environment, which may have far-reaching effects on not just development teams but through an entire organization.

- More time for innovation
- Better retention rates
- More revenue
- Business efficiency.

For example, a development environment with less manual tasks means that engineers can spend more time on revenue-generating projects. With less errors, teams are more effective and spend less time putting out fires. When processes, such as unit testing, are automated, engineers are happier and can concentrate on where they add the most value.
CI/CD gives automation into the DevOps lifecycle. With fewer manual work, DevOps teams work more effectively and with greater speed. An automated workflow also lessens the chance of human error and improves handoffs, which improves overall

operational efficiency. Organizations that execute CI/CD make better use of their resources and have a good edge over those who do not use CI/CD. (Continuous integration and delivery1, July 2020).

2.8.2 The importance of GitLab CI/CD

In order to complete all the required basics of CI/CD, many CI platforms depend on integrations with other tools to achieve those needs. Lots Of organizations have to keep on costly and complicated toolchains to have full CI/CD capabilities. This often means keeping on a separate SCM like Bitbucket or GitHub, connecting to a separate testing tool that connects to their CI tool, which connects to a deployment tool like Chef or Puppet, that also connects to different security and monitoring tools. Rather than just concentrating on building great software, organizations have to also keep and manage a complicated toolchain. GitLab is a single application for the full DevOps lifecycle, meaning we achieve all the basics for CI/CD in one environment. (Continuous integration and delivery1, July 2020).

2.8.3 The definition of Pipeline

A pipeline is a way to describe the set of stages placed in your CI/CD configuration. If your CI/CD has three stages: build, test, deploy, then your pipeline will look like this: Build → Test → Deploy

A CI/CD pipeline is the end to end steps your repository requires when new commits are pushed to the repository.
Finally, it was a long day. I read different sources to achieve my objectives. I think the outcomes of today were very good.

2.8.4 Friday 24 July 2020: Types of pipelines

Another wonderful day. I found it important to know the types of pipelines, so one of my goals for today will be to learn about the types of pipelines and the configuration of pipeline.
Pipelines are often configured in many various ways:

- Basic pipelines run the whole thing in each stage concurrently, afterward the next stage.

- Directed Acyclic Graph Pipeline (DAG) pipelines are depended on relationships between jobs and can run more rapidly than basic pipelines.
- Multi-project pipelines merge pipelines for various projects together.
- Parent-Child pipelines split up complex pipelines into one parent pipeline, which will trigger many child sub-pipelines, which all run within the same project and with an equivalent SHA.
- Pipelines for Merge Requests run merge requests only (rather than for each commit).
- Pipelines for Merged Results are merge request pipelines that act as if changes from the source branch have already been merged into the objective branch.
- Merge Trains utilize pipelines for merged results to queue merges one after the other.

2.8.5 Configure a pipeline

To configure a GitLab pipeline, we need to create a file called ".gitlab-ci.yml", this file contains the configurations of what CI does with your project. Its place in the root of your repository. On push an application to your repository, GitLab will search for the .gitlab-ci.yml file and begin jobs on Runners consistent with the contents of the file for that commit.

I can say that identifying the types of pipelines is very important to determine the type of pipeline required. Today's goals have been achieved by looking at the required sources, and the information obtained was important and new.

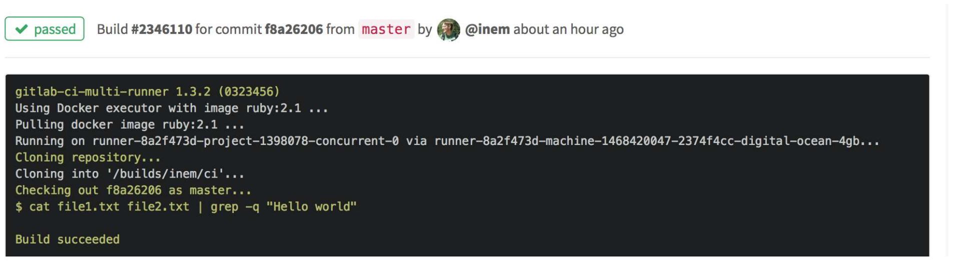
2.8.6 Saturday 25 July 2020: Creating a simple .gitlab-ci.yml file

I am very excited because today, I decided to create the first .gitlab-ci.yml file. I felt that it is time to go forward with this step after the days spent reading CI/CD pipeline and the level that I got.

Let's assume that we have two text files that contain the phrase 'Hello world,' and we want to run GitLab pipeline to test. In this case, we must add the following code to the .gitlab-ci.yml file as follows:

```
test:
  script: cat file1.txt file2.txt | grep -q 'Hello world'
```

Committing it, our build is successful, as shown in figure 57.



passed Build #2346110 for commit f8a26206 from master by @inem about an hour ago

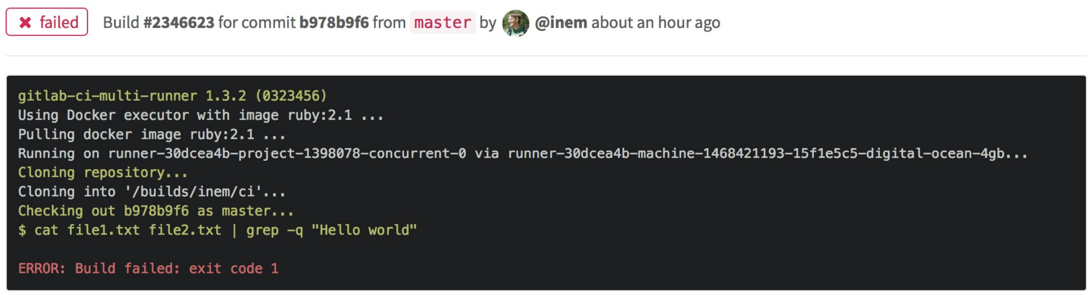
```
gitlab-ci-multi-runner 1.3.2 (0323456)
Using Docker executor with image ruby:2.1 ...
Pulling docker image ruby:2.1 ...
Running on runner-8a2f473d-project-1398078-concurrent-0 via runner-8a2f473d-machine-1468420047-2374f4cc-digital-ocean-4gb...
Cloning repository...
Cloning into '/builds/inem/ci'...
Checking out f8a26206 as master...
$ cat file1.txt file2.txt | grep -q "Hello world"

Build succeeded
```

Figure 57. The build is successful.

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

If we changed the phrase to 'Hello Africa' the build will fail, as shown in figure 58.

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/



failed Build #2346623 for commit b978b9f6 from master by @inem about an hour ago

```
gitlab-ci-multi-runner 1.3.2 (0323456)
Using Docker executor with image ruby:2.1 ...
Pulling docker image ruby:2.1 ...
Running on runner-30dcea4b-project-1398078-concurrent-0 via runner-30dcea4b-machine-1468421193-15f1e5c5-digital-ocean-4gb...
Cloning repository...
Cloning into '/builds/inem/ci'...
Checking out b978b9f6 as master...
$ cat file1.txt file2.txt | grep -q "Hello world"

ERROR: Build failed: exit code 1
```

Figure 58. The build is failed.

To run jobs sequentially, we will define the order by specifying stages, so our .gitlab-ci.yml file will be as follows:

```
“stages:
 - compile
 - test
 - package
compile:
 stage: compile
 script: cat file1.txt file2.txt > compiled.txt
 artifacts:
   paths:
   - compiled.txt
   - expire_in: 20 minutes
test:
 stage: test
 script: cat compiled.txt | grep -q 'Hello world'

package:
 stage: package
 script: cat compiled.txt | gzip > packaged.gz
 artifacts:
   paths:
   - packaged.gz”
```

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

If we do not want that "compile" file to be downloadable, we will make our temporary artifacts expire by setting expire_in to '20 minutes'.

By the end of the day, my goal was achieved successfully; with this example, I felt happy, and the competence gained during that day was wonderful.

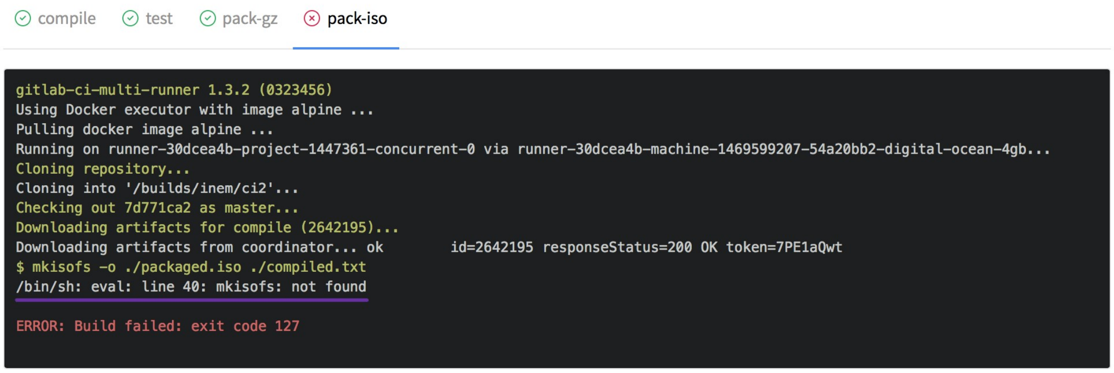
2.8.7 Sunday 26 July 2020: Dealing with complex scenarios

Today's goal is a continuation of yesterday, but today a more complex scenario than yesterday will be dealt with. Let's assume we want to package our app into .iso image instead of .gz; since CI does the whole work, we can add more jobs to it. ISO images can be created by using the mkisofs command. Here's how our config should look:

```
“image: alpine
stages:
 - compile
 - test
 - package
# ... "compile" and "test" jobs are skipped here for the sake of compactness
pack-gz:
 stage: package
 script: cat compiled.txt | gzip > packaged.gz
 artifacts:
   paths:
   - packaged.gz
pack-iso:
 stage: package
 script:
 - mkisofs -o ./packaged.iso ./compiled.txt
 artifacts:
   paths:
   - packaged.iso”
```

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

Note that it is not necessary that the job name should be the same. In fact, if they were the same, it is not possible to run the jobs in parallel inside the same stage. Anyhow, the build is failing; see figure 59.



```
gitlab-ci-multi-runner 1.3.2 (0323456)
Using Docker executor with image alpine ...
Pulling docker image alpine ...
Running on runner-30dcea4b-project-1447361-concurrent-0 via runner-30dcea4b-machine-1469599207-54a20bb2-digital-ocean-4gb...
Cloning repository...
Cloning into '/builds/inem/ci2'...
Checking out 7d771ca2 as master...
Downloading artifacts for compile (2642195)...
Downloading artifacts from coordinator... ok        id=2642195 responseStatus=200 OK token=7PE1aQwt
$ mkisofs -o ./packaged.iso ./compiled.txt
/bin/sh: eval: line 40: mkisofs: not found

ERROR: Build failed: exit code 127
```

Figure 59. The build is failed.

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

The problem is that mkisofs is not contained in the alpine image, so we need to install it first.

The goal of today has been achieved by applying the example in practice. Today, I got a lot of information that was very important and useful, and I think it is important to continue dealing with other cases in order to obtain an ideal pipeline.

2.8.8 Monday 27 July 2020: Dealing with missing software/packages

The goal of today is to continue building the pipeline in parallel. From my point of view, this is one of the most important types of pipelines, so I am very excited to apply it practically.

According to the Alpine Linux website, mkisofs is a part of the xorriso and cdrkit packages. These are the wonderful commands that we need to install a package:

```
“echo "ipv6" >> /etc/modules  # enable networking
apk update                # update packages list
apk add xorriso           # install package”
```

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

For CI, these are just the same as any other commands. The list of commands we want to pass to the script should look like this:

```
“script:
- echo "ipv6" >> /etc/modules
- apk update
- apk add xorriso
- mkisofs -o ./packaged.iso ./compiled.txt”
```

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

Let's put commands concerning package installation in before_script. Note that if you use before_script at the beginning of a configuration, then the commands will run before all jobs. In our case, we just want to run it before a specific job.

Our final .gitlab-ci.yml:

```
“image: alpine
stages:
  - compile
  - test
  - package
compile:
  stage: compile
  script: cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
    - compiled.txt
    expire_in: 20 minutes
test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'
pack-gz:
  stage: package
  script: cat compiled.txt | gzip > packaged.gz
  artifacts:
    paths:
    - packaged.gz
pack-iso:
  stage: package
  before_script:
  - echo "ipv6" >> /etc/modules
  - apk update
  - apk add xorriso
  script:
  - mkisofs -o ./packaged.iso ./compiled.txt
  artifacts:
    paths:
    - packaged.iso”
```

Now, we have three sequential stages, but jobs pack-gz and pack-iso, within the package stage, are running in parallel, see figure 60.

source: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

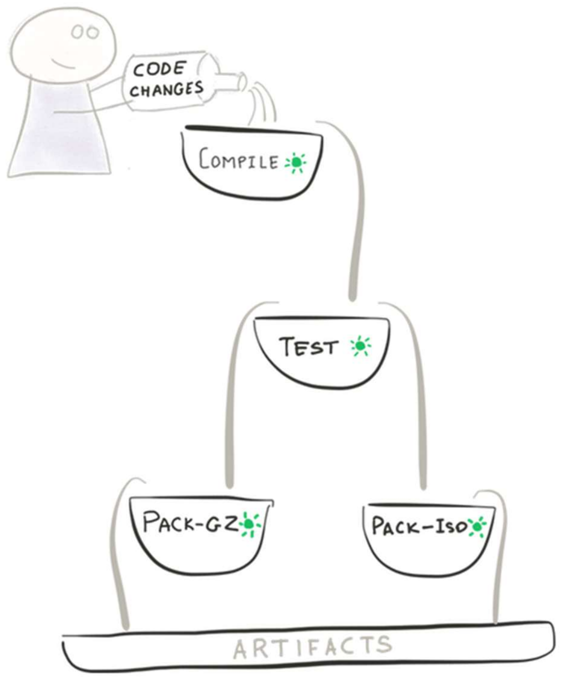


Figure 60. The jobs are running in parallel.

With the result of this example, I believe that the goal assigned to this day was achieved by creating an ideal pipeline, and with this result, I see that the information or rather the efficiency gained for this day was great and very impressive.

2.8.9 Weekly Analysis

During this week, I have been getting familiar with the most important benefits of CI/CD which, I had to know them before starting it practically. Then I addressed the importance of GitLab CI/CD. Besides, I found it is important to define the pipeline, especially.

Through this week, I also familiarized types of pipelines, which was totally new for me, and knowing about these types has increased my knowledge of what a CI/CD pipeline is.

Then I touched on how to configure a pipeline and it is the core of the research. Where I learned everything related to it. Afterward, I have created the first simple .gitlab-ci.yml. This file contains the configurations of what CI does with your project.

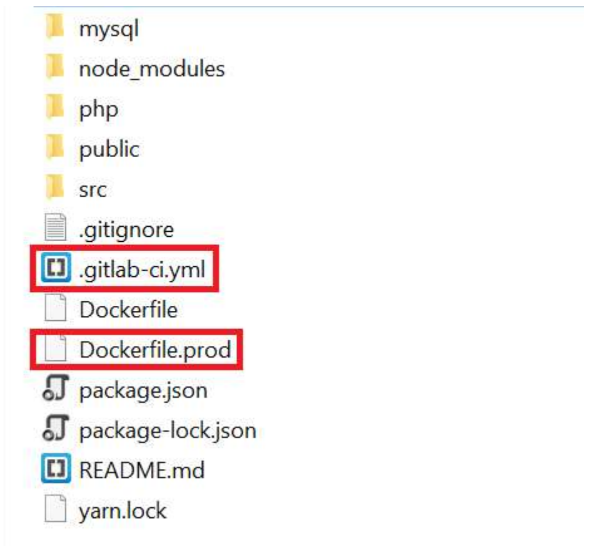
After creating this simple file, I found that it is important to create a file that simulates a scenario more complex. Then I created a more complex scenario until I created an integrated pipeline that works in parallel.

I think the results I gained during this week were very productive, and generally, I got a good level of new technologies, which increased my skills.

2.9 Week 9

2.9.1 Monday 14 December 2020: Dockerizing the application

Today, I decided to start creating the pipeline for the application I wanted to test. Today my objective will be the first stage of this pipeline, which is Docker. During the research period, I found two types of Dockerfile, a development type, and a production type. First, I created a file called .gitlab-ci.yml in the root of the application folder. Besides, a file called Dockerfile.prod, as shown in figure 61.



```
mysql
node_modules
php
public
src
.gitignore
.gitlab-ci.yml
Dockerfile
Dockerfile.prod
package.json
package-lock.json
README.md
yarn.lock
```

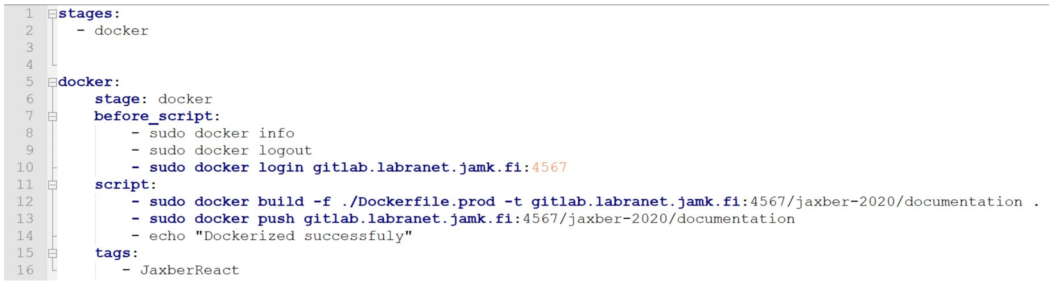
Figure 61. Application files

In the Dockerfile.prod I copied and installed all the required files and dependencies to create the Docker image, as shown in figure 62.

```
# build environment
FROM node:13.12.0-alpine as build
# FROM saadstj/node:13.12.0-alpine as build
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN npm ci --silent
RUN npm install react-scripts@3.4.1 -g --silent
COPY . ./
RUN npm run build

# production environment
FROM nginx:stable-alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Figure 62. Docker.prod file

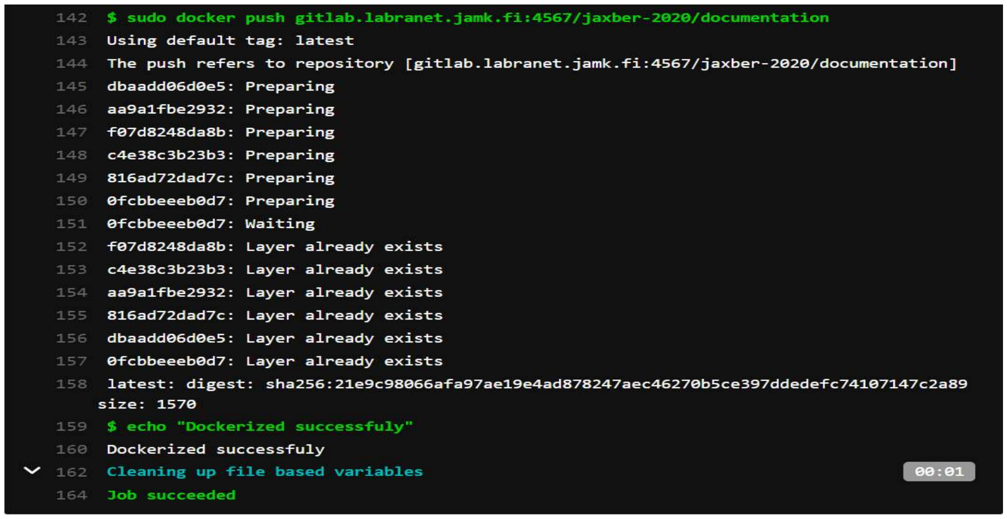
Then in the .gitlab-ci.yml file, I defined the stages and script. To start creating the Docker image of the application and push it into the container registry, as shown in figure 63.



```
stages:
  - docker


docker:
    stage: docker
    before_script:
        - sudo docker info
        - sudo docker logout
        - sudo docker login gitlab.labranet.jamk.fi:4567
    script:
        - sudo docker build -f ./Dockerfile.prod -t gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation .
        - sudo docker push gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation
        - echo "Dockerized successfuly"
    tags:
       - JaxberReact
```

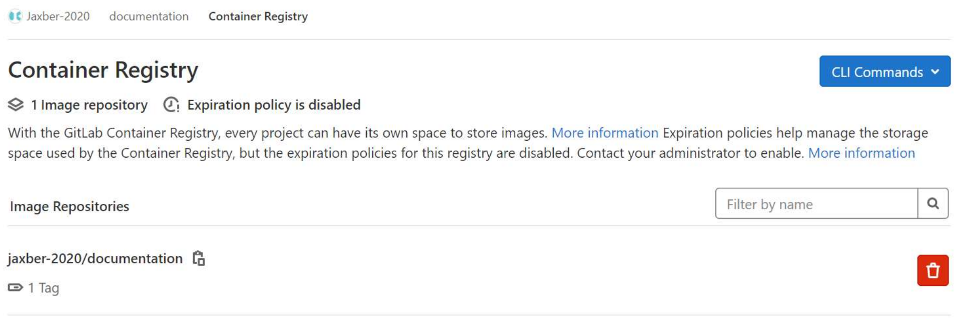
Figure 63. The .gitlab-ci.yml file

After I committed all the files into the GitLab repository, the CI/CD pipeline run and started creating the image, then pushed the image into the container registry, as shown in figures 64 and 65.



```
$ sudo docker push gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation
Using default tag: latest
The push refers to repository [gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation]
dbaadd06d0e5: Preparing
aa9a1fbe2932: Preparing
f07d8248da8b: Preparing
c4e38c3b23b3: Preparing
816ad72dad7c: Preparing
0fcbbeeeb0d7: Preparing
0fcbbeeeb0d7: Waiting
f07d8248da8b: Layer already exists
c4e38c3b23b3: Layer already exists
aa9a1fbe2932: Layer already exists
816ad72dad7c: Layer already exists
dbaadd06d0e5: Layer already exists
0fcbbeeeb0d7: Layer already exists
latest: digest: sha256:21e9c98066afa97ae19e4ad878247aec46270b5ce397ddedefc74107147c2a89
size: 1570
$ echo "Dockerized successfuly"
Dockerized successfuly
Cleaning up file based variables                                                00:01
Job succeeded
```

Figure 64. Docker image created and pushed successfully.



Jaxber-2020 documentation Container Registry

Container Registry

CLI Commands

1 Image repository Expiration policy is disabled

With the GitLab Container Registry, every project can have its own space to store images. More information Expiration policies help manage the storage space used by the Container Registry, but the expiration policies for this registry are disabled. Contact your administrator to enable. More information

Image Repositories

Filter by name

jaxber-2020/documentation

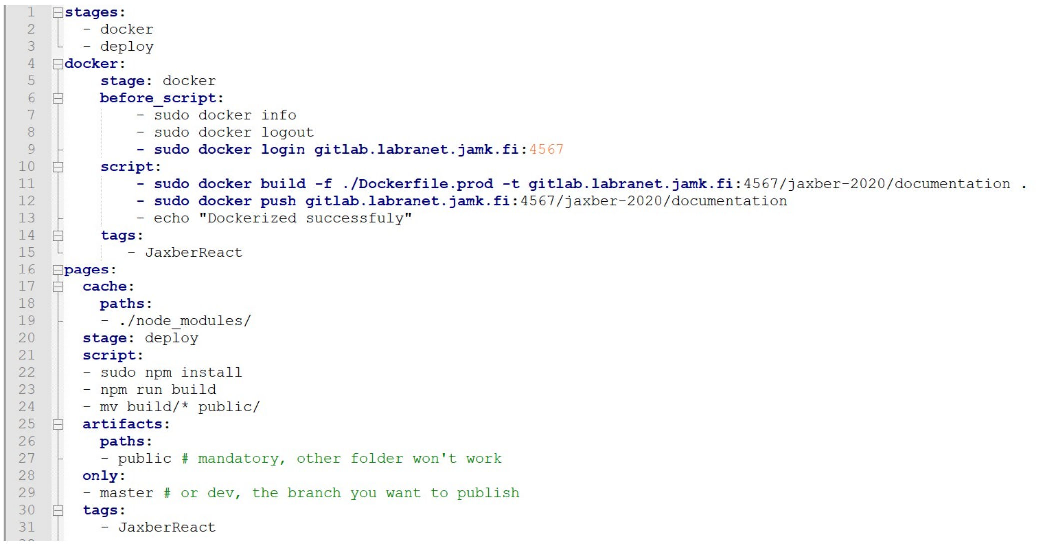
1 Tag

Figure 65. The pushed image in the container registry.

The goal of today is achieved successfully by applying the example in practice. Today, I got a lot of information that was very important and useful. Also, the error message that I faced during the process of creating the Docker image helped me increase my skills to avoid this error in the future.

2.9.2 Tuesday 15 December 2020: GitLab pages

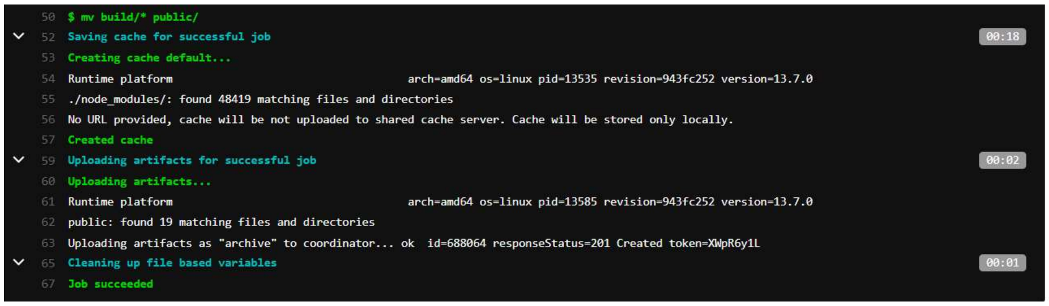
Today, I will address one of the most important GitLab features, which is GitLab pages. This feature helps to deploy the application, which will facilitate creating test automation for this application. Therefore, my objectives that I set for today are GitLab pages and test automation. I defined the second stage in the .gitlab-ci.yml file and became, as shown in figure 66.



```yaml
stages:
  - docker
  - deploy
docker:
    stage: docker
    before_script:
        - sudo docker info
        - sudo docker logout
        - sudo docker login gitlab.labranet.jamk.fi:4567
    script:
        - sudo docker build -f ./Dockerfile.prod -t gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation .
        - sudo docker push gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation
        - echo "Dockerized successfuly"
    tags:
       - JaxberReact
pages:
  cache:
    paths:
    - ./node_modules/
  stage: deploy
  script:
  - sudo npm install
  - npm run build
  - mv build/* public/
  artifacts:
    paths:
    - public # mandatory, other folder won't work
  only:
  - master # or dev, the branch you want to publish
  tags:
    - JaxberReact
```

Figure 66. The .gitlab-ci.yml file after adding the second stage.

It is important to say that I had to set the path to the public folder because another folder will not work after I committed the new changes. The job succeeded by deploying the application, as shown in figures 67 and 68.



```
$ mv build/* public/
Saving cache for successful job                                                                                00:18
Creating cache default...
Runtime platform                                   arch=amd64 os=linux pid=13535 revision=943fc252 version=13.7.0
./node_modules/: found 48419 matching files and directories
No URL provided, cache will be not uploaded to shared cache server. Cache will be stored only locally.
Created cache
Uploading artifacts for successful job                                                                         00:02
Uploading artifacts...
Runtime platform                                   arch=amd64 os=linux pid=13585 revision=943fc252 version=13.7.0
public: found 19 matching files and directories
Uploading artifacts as "archive" to coordinator... ok  id=688064 responseStatus=201 Created token=XWpR6y1L
Cleaning up file based variables                                                                               00:01
Job succeeded
```

Figure 67. Job succeeded.

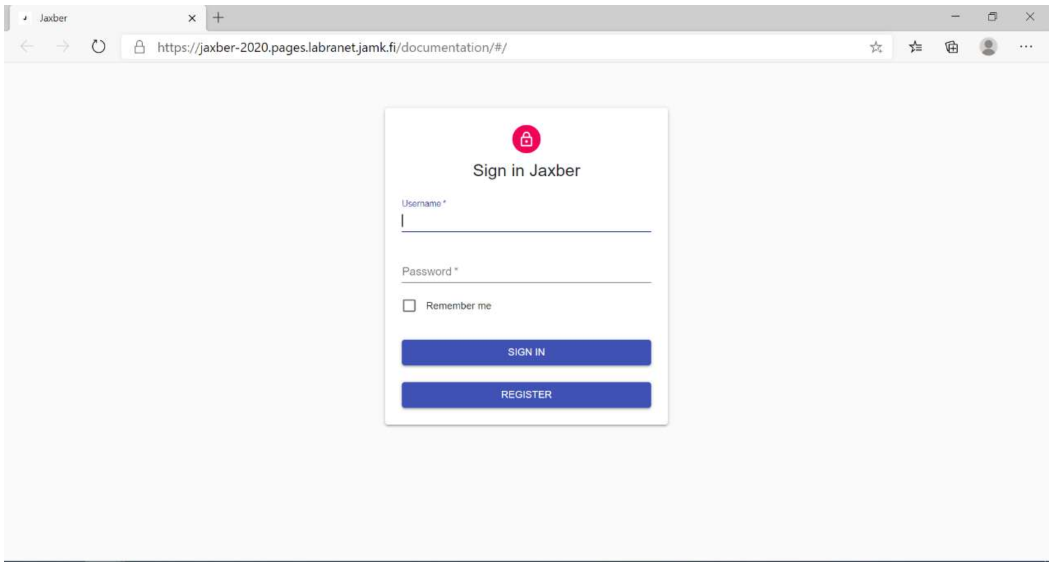
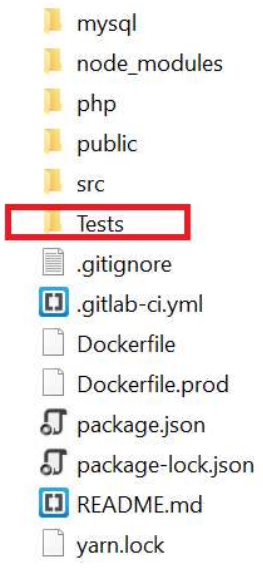


Figure 68. The application after deploying on GitLab pages.

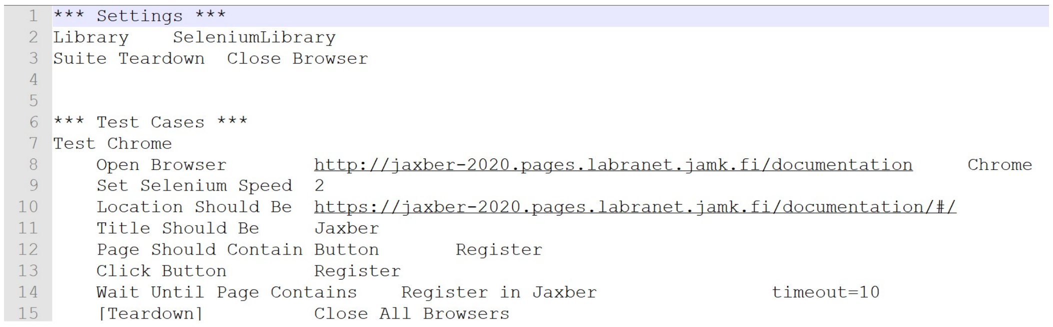
2.9.3 Test automation

This will be the third stage in the pipeline. In this stage, I created a folder in the root of the application called Tests containing the test cases file, written using the Selenium library, as shown in figures 69, 70, and 71.



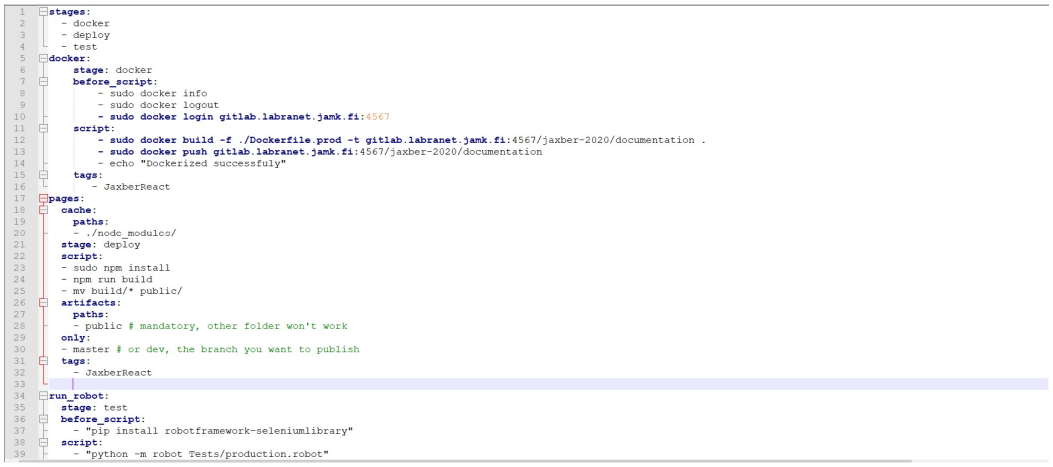
mysql
node_modules
php
public
src
Tests
.gitignore
.gitlab-ci.yml
Dockerfile
Dockerfile.prod
package.json
package-lock.json
README.md
yarn.lock

Figure 69. Application files



```
*** Settings ***
Library     SeleniumLibrary
Suite Teardown  Close Browser


*** Test Cases ***
Test Chrome
    Open Browser        http://jaxber-2020.pages.labranet.jamk.fi/documentation     Chrome
    Set Selenium Speed  2
    Location Should Be  https://jaxber-2020.pages.labranet.jamk.fi/documentation/#/
    Title Should Be     Jaxber
    Page Should Contain Button       Register
    Click Button        Register
    Wait Until Page Contains    Register in Jaxber                timeout=10
    [Teardown]          Close All Browsers
```

Figure 70. Test cases file.



```
stages:
  - docker
  - deploy
  - test
docker:
    stage: docker
    before_script:
        - sudo docker info
        - sudo docker logout
        - sudo docker login gitlab.labranet.jamk.fi:4567
    script:
        - sudo docker build -f ./Dockerfile.prod -t gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation .
        - sudo docker push gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation
        - echo "Dockerized successfuly"
    tags:
       - JaxberReact
pages:
  cache:
    paths:
    - ./node_modules/
  stage: deploy
  script:
  - sudo npm install
  - npm run build
  - mv build/* public/
  artifacts:
    paths:
    - public # mandatory, other folder won't work
  only:
  - master # or dev, the branch you want to publish
  tags:
    - JaxberReact

run_robot:
  stage: test
  before_script:
    - "pip install robotframework-seleniumlibrary"
  script:
    - "python -m robot Tests/production.robot"
```

Figure 71. The final shape of the .gitlab-ci.yml file after adding the third stage.

After I pushed the new changes, the pipeline became three stages, and the test passed successfully, as in figures 72 and 73.

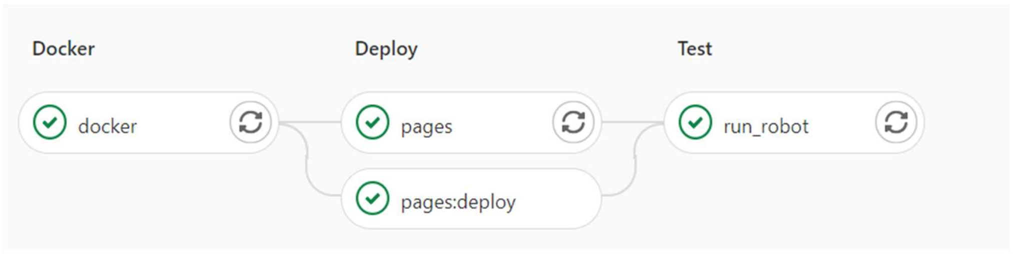
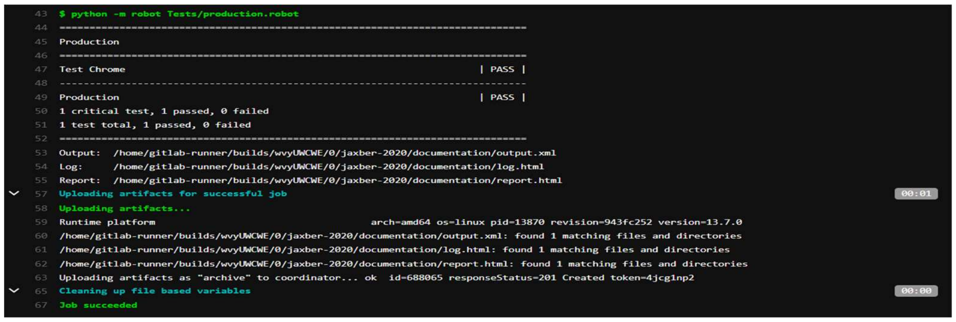


Figure 72. The pipeline stages.



```
$ python -m robot Tests/production.robot
==============================================================================
Production
==============================================================================
Test Chrome                                                           | PASS |
------------------------------------------------------------------------------
Production                                                            | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/output.xml
Log:     /home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/log.html
Report:  /home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/report.html
Uploading artifacts for successful job                                                              00:01
Uploading artifacts...
Runtime platform                                    arch=amd64 os=linux pid=13870 revision=943fc252 version=13.7.0
/home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/output.xml: found 1 matching files and directories
/home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/log.html: found 1 matching files and directories
/home/gitlab-runner/builds/wvyUWCWE/0/jaxber-2020/documentation/report.html: found 1 matching files and directories
Uploading artifacts as "archive" to coordinator... ok  id=688065 responseStatus=201 Created token=4jcg1np2
Cleaning up file based variables                                                                    00:00
Job succeeded
```

Figure 73. Test with Robot passed successfully.

The day was interesting, where I achieved the objectives I set for today, which helped me to know more about GitLab features and test automation. By the end of these goals, I can say that I became familiar with creating a GitLab pipeline.

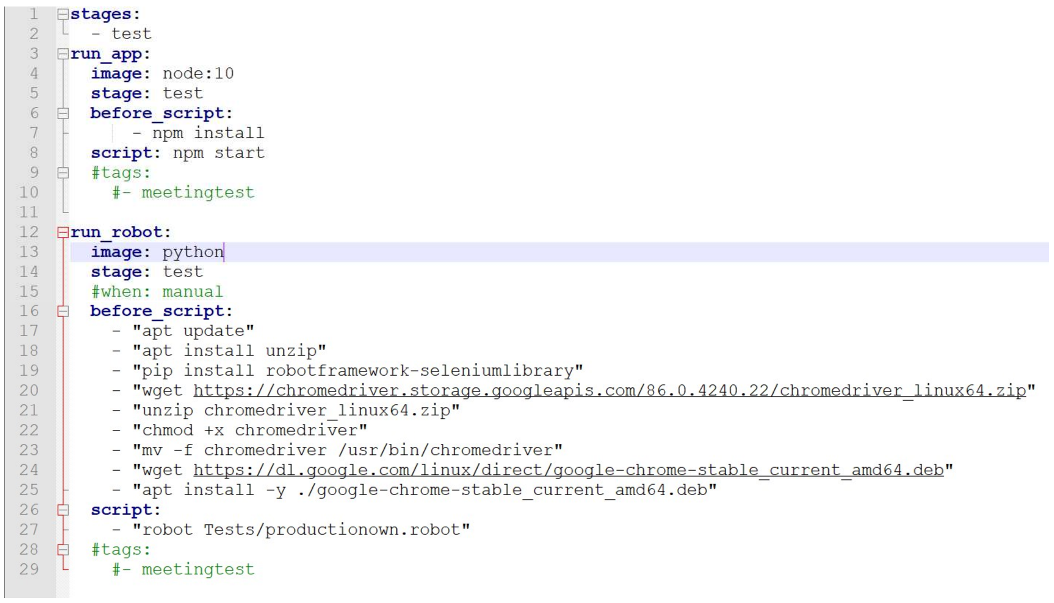
2.9.4 Some Problems I faced

During the Docker stage, I faced the following error:

```
$ sudo docker build -f ./Dockerfile.prod -t gitlab.labranet.jamk.fi:4567/jaxber-2020/documentation .
Step 1 : FROM node:13.12.0-alpine as build
Error parsing reference: "node:13.12.0-alpine as build" is not a valid repository/tag
```

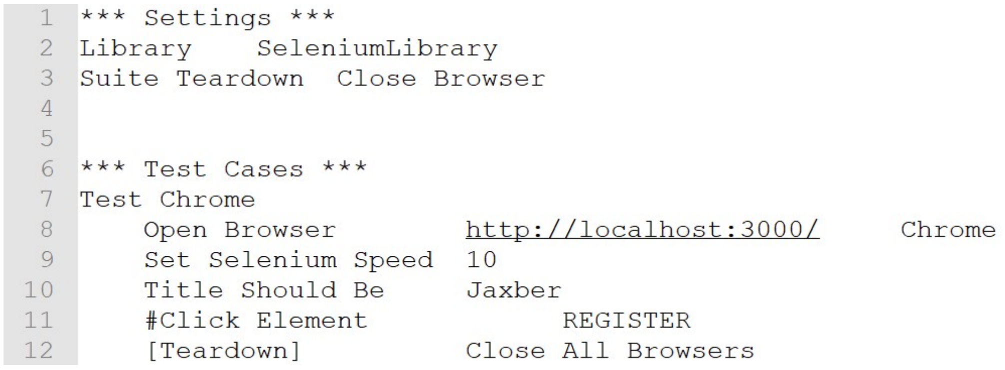
This problem is a result of the variable “as build” so in order to solve it, you have to install the latest version of the Docker engine on your gitlab-runner.

Related to Robot Framework stage, I created a stage running in the parallel type of pipeline. The first one is to run the application, and the second one to run the file containing test cases of Robot Framework. In the test case file, I tried to test the application depending on the localhost. That was the problem I faced. The application was running, and the Robot was working faster than the application's running. Then the test failed, as shown in figures 74, 75, and 76.



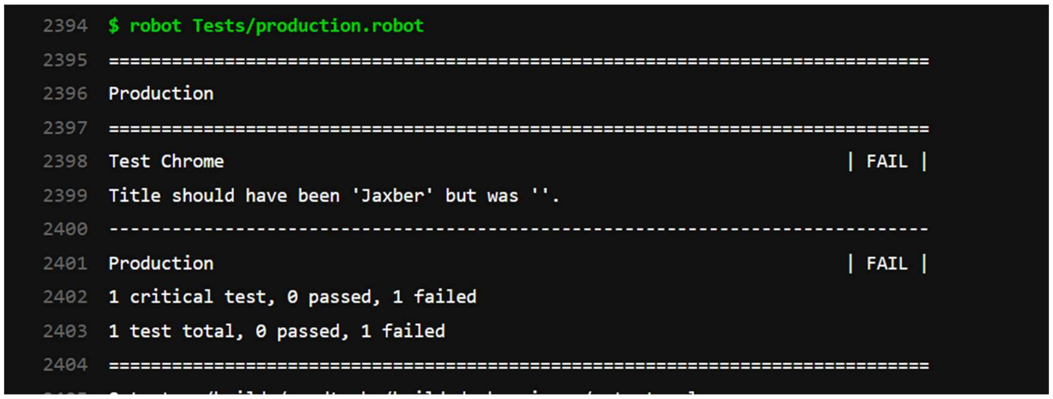
```yaml
stages:
  - test
run_app:
  image: node:10
  stage: test
  before_script:
      - npm install
  script: npm start
  #tags:
    #- meetingtest

run_robot:
  image: python
  stage: test
  #when: manual
  before_script:
    - "apt update"
    - "apt install unzip"
    - "pip install robotframework-seleniumlibrary"
    - "wget https://chromedriver.storage.googleapis.com/86.0.4240.22/chromedriver_linux64.zip"
    - "unzip chromedriver_linux64.zip"
    - "chmod +x chromedriver"
    - "mv -f chromedriver /usr/bin/chromedriver"
    - "wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb"
    - "apt install -y ./google-chrome-stable_current_amd64.deb"
  script:
    - "robot Tests/productionown.robot"
  #tags:
    #- meetingtest
```

Figure 74. The .gitlab-ci.yml file



```
*** Settings ***
Library    SeleniumLibrary
Suite Teardown  Close Browser


*** Test Cases ***
Test Chrome
    Open Browser          http://localhost:3000/     Chrome
    Set Selenium Speed    10
    Title Should Be       Jaxber
    #Click Element              REGISTER
    [Teardown]            Close All Browsers
```

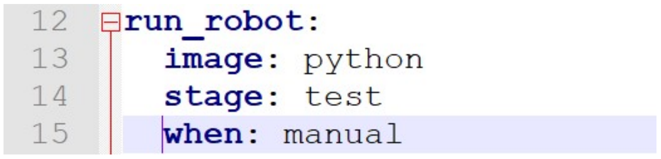
Figure 75. The test cases file



```
$ robot Tests/production.robot
==============================================================================
Production
==============================================================================
Test Chrome                                                           | FAIL |
Title should have been 'Jaxber' but was ''.
------------------------------------------------------------------------------
Production                                                            | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
==============================================================================
```

Figure 76. Failing the test

I tried to solve this problem by reducing the Selenium Speed but that did not help. I also tried to overcome this problem by setting Robot's operation to run manually, as shown in figure 77.



```
run_robot:
  image: python
  stage: test
  when: manual
```

Figure 77. Robot runs manually.

The solutions I mentioned above were not helpful. I had to search for a long time until I found the answer. The problem was that the localhost works on a personal computer and can not be considered as a URL. To solve this problem, I have to deploy the application on one of the host websites. Fortunately, GitLab provides this service, so I can deploy the application to the pages of GitLab, then I can test the application using Robot Framework as I mentioned in 2.9.2 and 2.9.3.

2.9.5 Weekly Analysis

During this week, I became familiar with creating a pipeline. Where I learned about dockerizing an application for production. Also, this week I learned a lot about the GitLab features, especially GitLab Pages. Where this feature helps to deploy an application then facilitates the test automation.

Also, this week, I touched on creating a test cases file using the Selenium library in order to test an application using Robot Framework. I can also say that the problems I faced during creating this pipeline and its solutions increased my ability to avoid them in the future, thus shortening the time. Finally, the goal of the thesis, which is to create an environment development on GitLab to test the company's application by creating a pipeline using Robot Framework, was achieved successfully.

3. Discussion

3.1 Research Results

The research goal, which was to create a modern software development environment for the cloud-based application called Jaxber, was achieved. That includes continuous integration, deployment (CI/CD), and automated testing, made this research very informative. During the search online, I found that learning new technologies like Docker, Git, and Robot Framework became very important. Indeed, some companies require knowing these technologies as a condition to get the job.

For me, everything was totally new except my knowledge of GitLab's basics. However, it should be emphasized that great strides have been made in the learning of these technologies as well. I will now explain the results I achieved during the thesis writing process. I believe that the created environment works successfully.

First, I created a pipeline including all the technologies addressed in this research. The first stage was about the Docker file. Where I managed to dockerize "which means the process of running an application within a Docker container" the application for production. This stage's success made me feel I can dockerize an application for development and production. Even the errors I got during this stage made my skills better to avoid these errors in the future.

In the second stage, I consider the core of the research. I managed to create test automation using Robot Framework. This stage was the most difficult. It took a lot of searching and communicating with experts. Until I got a new GitLab feature, which is called Pages. This feature helped in deploying the application and provided a URL. Thus, I managed to create the test automation. But I had to add the deploy stage to the pipeline, where the pipeline splits into three stages, which are docker, deploy (on GitLab Pages), and the test (Robot Framework). These results can be used by the students, companies, or software engineers interested in DevOps or test automation. Although the results are successful, more test cases should add to the test file.

It is important to say that I created my virtual machine, where I installed a GitLab runner and the most important packages required to reduce the time of the test.

3.2 Recommendations

This research for everybody who is interested in DevOps, students, and software engineers. Through this research, the reader can understand modern software technologies—for example, Docker, Robot Framework, and pipeline. The research results help the reader in shortening the time and effort to learn new technologies. The assigner asked me to attend the conference AWS DevDay Online which was held on October, 15 and give recommendations on how to develop Jaxber development environment in the near future. I attended the following lectures:

- Architecting and Running Microservices
- Container Cluster Optimisation
- Getting started with CICD for
- Implementing Cloud Security Automation at scale
- modern application development.

This conference was more than wonderful, where I learned a lot, especially in relation to my research topic of pipeline creation.
In my opinion, I recommend creating a development environment on AWS because you will get the control and confidence you need to securely run your business with the most secure cloud computing environment. Moreover, cloud-based computing has economic benefits as users would pay for only the services they request. (AWS Cloud Security, February 2021)
The reliability and safety of AWS may be safer than GitLab environment, according to AWS. In terms of Integrations AWS CodeCommit can connect with the following products:

- AWS Management Console
- AWS CLI
- AWS SDKs.

AWS CodeCommit also supports all Git commands and your existing Git tools. Also, you can integrate with your development environment plugins and continuous integration/continuous delivery systems.

The results obtained are related to web-based applications only. It is not applicable to all kinds of applications. For example, I addressed Jaxber React version application as I built it to the web by running the web build script then tested it using SeleniumLibrary.

In the case of React Native, we need to use AppiumLibrary to test the app instead of SeleniumLibrary. So, I recommend the new researchers to read and understand the differences between Robot Framework libraries.

3.3 Reflections on Research Method

“A poorly designed diary study can involve considerable effort but may yield little useful information.” (Bolger, Davis & Rafaeli 2003, 581.)

“Novice student autoethnographers also face considerable difficulties with the research, thesis production, examination, and supervision process.” (Doloriert & Sambrook 2012, 88.)

These warnings should be taken seriously but they need not be feared. The thesis writing process has been demanding but also satisfying. When I started doing the research diary my plan was to write a page or two pages per day for nine weeks. By the middle of the reporting period, I noticed that there was already plenty of research material. The ease of writing was clearly one of the best aspects of the method.
Weekly analyses were very important. It summarizes what I learned for the last period. It also reveals the competence of the information that I gained, and identified the points of strength and weakness that made me plan better for the upcoming weeks.

3.4 Reflections on the Research

I found the research is important and useful for any new researcher in this field. This was one of the motivations to go through this research. This research did not cover all the details related to the aspects I addressed because it is so wide. But I can consider it as a solid basis for new researchers or for those who are interested in learning modern technologies.
During the research, I gained a lot of new information regarding Docker, git, and Robot Framework, which increased my skills in this field. This research required a lot of work and effort, where I read and watched different kinds of sources and videos.

References

AWS Cloud Security, Accessed in February 2021. Retrieved from: https://aws.amazon.com/security/

Bolger, N., Davis, A., & Rafaeli, E. 2003. Diary Methods: Capturing Life as it is Lived. Annual Review of Psychology, 579-616. Accessed on 3 August 2020. Retrieved from: https://www.researchgate.net/publication/10974933_Diary_Methods_Capturing_Life_as_it_is_Lived

CI/CD pipelines. Accessed in July 2020. Retrieved from: https://docs.gitlab.com/ee/ci/pipelines/

Containers. Accessed in July 2020. Retrieved from: https://www.ibm.com/cloud/learn/containers#toc-containeri-Qt_5RXC_

Continuous integration and delivery. Accessed in July 2020. Retrieved from: https://about.gitlab.com/ci-cd/

Continuous integration and delivery1, Accessed in June 2020. Retrieved from: https://about.staging.gitlab.com/ci-cd/

Docker Image vs Container: The Major Differences. Accessed in June 2020. Retrieved from: https://phoenixnap.com/kb/docker-image-vs-container

Docker overview. Accessed in June 2020. Retrieved from: https://docs.docker.com/get-started/overview/

Dockerfile reference. Accessed in June 2020. Retrieved from: https://docs.docker.com/engine/reference/builder/

GitLab CI: Run jobs sequentially, in parallel or build a custom pipeline. Accessed in July 2020. Retrieved from: https://about.gitlab.com/blog/2016/07/29/the-basics-of-gitlab-ci/

GitLab vs GitHub: Key differences & similarities. Accessed in July 2020. Retrieved from: https://usersnap.com/blog/gitlab-github/

GitLab. Accessed in July 2020. Retrieved from: https://en.wikipedia.org/wiki/GitLab

How is Docker different from a virtual machine?. Accessed in June 2020. Retrieved from: https://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-virtual-machine

Lischetzke, T. (2014). Daily diary methodology. Accessed on 3 August 2020. Retrieved from: https://www.researchgate.net/publication/260917855_Daily_Diary_Methodology

Robot — Working With Browsers Using Selenium Library. Accessed in July 2020. Retrieved from: https://www.tutorialspoint.com/robot_framework/robot_framework_tutorial.pdf#page=73&zoom=100,42,1

Robot Framework - Introduction to Ride. Accessed in July 2020. Retrieved from: https://www.tutorialspoint.com/robot_framework/robot_framework_introduction_ride.htm

Robot Framework - Robot Framework Tutorial. Accessed in July 2020. Retrieved from: https://www.tutorialspoint.com/robot_framework/index.htm

Robot Framework User Guide. Accessed in July 2020. Retrieved from: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Robot Framework, Working With Setup And Teardown. Accessed in July 2020. Retrieved from: https://www.tutorialspoint.com/robot_framework/robot_framework_tutorial.pdf#page=176&zoom=100,42,1

Robot Framework. Accessed in July 2020. Retrieved from: https://robotframework.org/

Sumit Bisht, October 2013. Robot Framework Test Automation. Accessed in July 2020. Retrieved from: https://subscription.packtpub.com/book/application_development/9781783283033

What is Docker, Accessed in June 2020. Retrieved from: https://3alam.pro/3mmarg97/series/introduction-to-docker/lessons/what-is-docker

What is the difference between “expose” and “publish” in Docker?. Accessed in June 2020. Retrieved from: https://stackoverflow.com/questions/22111060/what-is-the-difference-between-expose-and-publish-in-docker/22150099#22150099