Valtteri Kurhela

Automation of virtualized hardware test environment

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor’s Thesis

14 May 2021

Abstract

Author:	Valtteri Kurhela
Title:	Automation of virtualized hardware test environment
Number of Pages:	24 pages
Date:	14 May 2021

Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Smart Systems
Supervisors:	Joseph Hotchkiss, Senior Lecturer

This thesis aimed to research test automation and how it could be used in a virtualized hardware test environment to deploy a test automation script. The motive for this thesis was to investigate options to decrease repetitive manual work for BTS O&M software developers.

The first phase of the thesis was to research virtualization, study Radio Access Networks, familiarize myself with Gerrit, Jenkins, and Robot Framework.

The thesis explains the state of the current virtualized hardware test server and tools that enable test automation deployment. The thesis will go through the deployment steps of previous test automation system implementation, as the deployment steps are the same. A new automation system is conceptualized and determined if such a concept would be able to be implemented.

The thesis explains the basics and capabilities of 5G and what BTS O&M does in the 5G field. The thesis will also explain the basics of virtualization and the difference between virtual machines and containers.

Based on this study done for this thesis, it can be concluded that automation of test environments reduces the daily workload of software developers. The research concludes, that the conceptualized automation system would be able to be implemented on top of the current virtualized hardware test environment.

Keywords: Test Automation, RAN, Virtualization, Robot Framework, 5G

Tiivistelmä

Tekijä:	Valtteri Kurhela
Otsikko:	Automation of virtualized hardware test environment
Sivumäärä:	24 sivua
Aika:	14.5.2021

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Smart Systems
Ohjaajat:	Joseph Hotchkiss, Lehtori

Tämän opinnäytetyön tavoitteena oli tutkia testausautomaatiota ja miten sitä voitaisiin käyttää virtualiisoidussa laitteistotestiympäristössä. Opinnäytetyön motiivina oli tutkia vaihtoehtoja vähentämään toistuvaa manuaalista työtä BTS O&M -ohjelmistokehittäjille.

Opinnäytetyön ensimmäinen vaihe oli tutkia virtualisointia ja radioliitäntäverkkoja sekä tutustua Gerritiin, Jenkinsiin ja Robot Frameworkiin.

Opinnäytetyössä selitetään nykyisen virtualisoidun laitteistotestipalvelimen tila ja työkalut, jotka mahdollistavat testausautomaation käyttöönoton. Opinnäytetyö kuvaa edellisen testausautomaatiojärjestelmän käyttöönoton vaiheet, koska käyttöönottovaiheet ovat samat nykyäänkin. Uusi automaatiojärjestelmä on teorisoitu ja lisäksi selvitetty, onko sen toteuttaminen mahdollista myös käytännössä.

Opinnäytetyössä selitetään 5G:n perusteet ja ominaisuudet sekä mitä BTS O&M tekee 5G-toimialalla. Opinnäytetyössä selitetään myös virtualisaation perusteet ja miten virtuaalikoneet eroavat säiliöistä.

Tämän opinnäytetyön perusteella voidaan päätellä, että testausympäristön automatisointi vähentää ohjelmistokehittäjien päivittäistä työmäärää. Opinnäytetyössä osoitetaan, että teorisoitu automaatiojärjestelmä voidaan toteuttaa nykyisen virtualisoidun laitteistoympäristön päälle.

Avainsanat: Testiautomaatio, RAN, Virtualisaatio, Robot Framework, 5G

Contents

List of Abbreviations

IDE	Integrated Development Environment
SSH	Secure Shell Protocol.
CU	Centralized Unit
DU	Distributed Unit
vCU	Virtualized Centralized Unit.
vDU	Virtualized Distributed Unit
5G	5th generation cellular mobile network.
BTS	Base Transceiver Station
O&M	Operations and Maintenance
MN	Mobile Networks
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
RIDE	Robot Integrated Development Environment
RAN	Radio Access Network
vRAN	Virtualized Radio Access Network
eMBB	Enhanced Mobile Broadband
URLLC	Ultra-Reliable Low Latency Communications

mMTC	Massive Machine Type Communications
LTE	Long Term Evolution
OS	Operating System

1 Introduction

Automation is at its all-time peak of popularity in modern software development teams because it allows tasks to be executed faster and more reliably than a human would.

Base Transceiver Station (BTS) Operations and Maintenance (O&M) team in Nokia Mobile Networks (MN) has a virtualized hardware test environment in use that lacks automation. The main task for this thesis is to make a concept of an automation system or a script that would allow software developers to test their code changes in the virtualized hardware test environment with the least amount of manual steps necessary. A system of this kind requires lots of background research and expertise for it to be deployed successfully.

The thesis studies the tools and deployment steps that are necessary for an automation system of this kind. The thesis also explains the basic concepts of virtualization, 5G and O&M. The thesis covers the surface of virtualized Radio Access Network (vRAN) and its sub-units that are running in the virtualized hardware test server.

The motivation for this task was to reduce the workload of software developers in BTS O&M that is caused by manual configuration steps. Requiring software developers to do the same manual configuration steps multiple times is an unnecessary workload that can and should be automated.

2 5G

2.1 5G explained

5G is the 5th generation cellular networking technology and the wireless telecommunication standard following 1G, 2G, 3G, and 4G. 5G is currently the newest generation of cellular networking that is in public use and it is one of the largest trends of the technology industry.

A cellular network or mobile network refers to a radio network, where the link between end devices is wireless. End devices communicate with each other by using radiowaves. End devices send radio waves to BTS which transmits them to other end devices or outer networks. A geographical area with wireless devices is called a cell. One cell has one BTS that end devices are connected to.

5G connectivity is a major feature for mobile phone manufacturers. Phones must have integrated hardware that enables them to be connected to 5G networks. 5G networks in Finland are currently in use in large cities such as Helsinki and Tampere, but they are not widespread in more rural areas.

2.2 5G capabilities

5G offers improved capabilities from 4G and 4G Long Term Evolution (LTE). 5G capabilities can be divided into three distinct sectors as seen in the figure below.

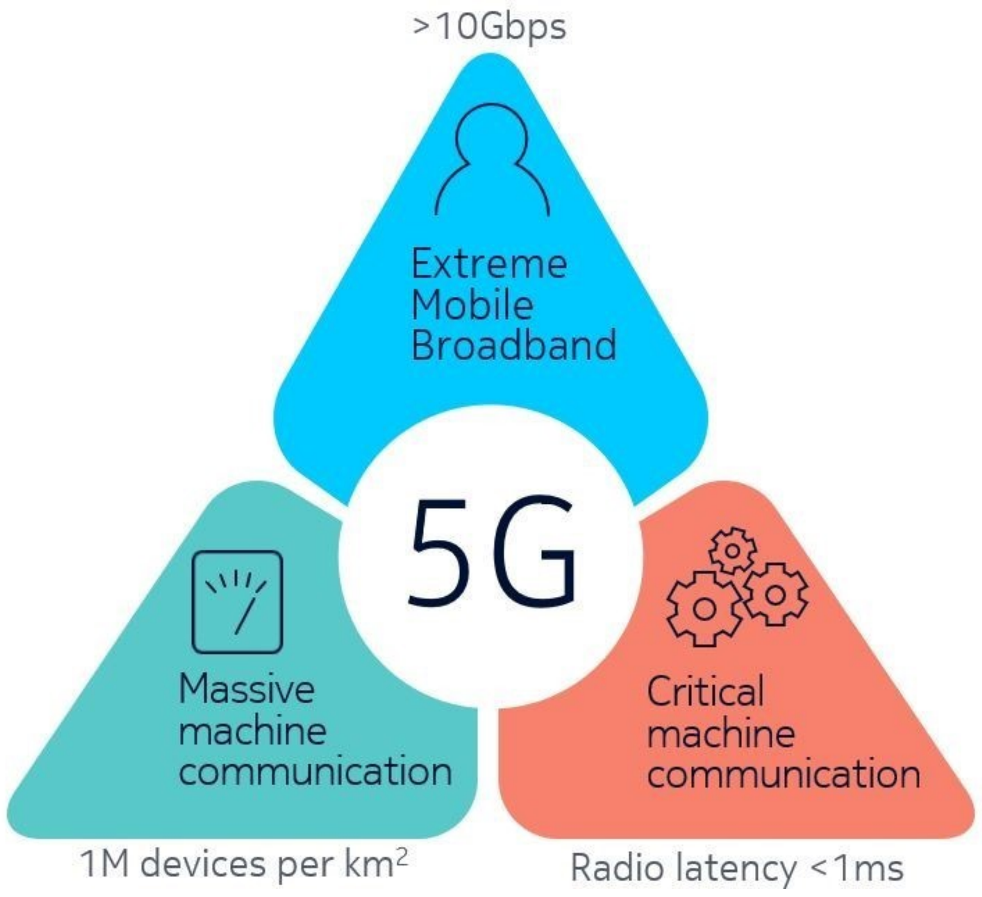


Figure 1: 5G capabilities triangle [1]

These capabilities are Extreme Mobile Broadband (eMBB), Ultra-reliable Low Latency Communication (URLLC), and Massive Machine Type Communications (mMTC) [1]

eMBB in 5G refers to the over 10 Gigabits per second transmission speed of data. This enables high-resolution streaming of movies over the internet or making cloud gaming a possibility. Video games require powerful and expensive hardware, thus making the hobby limited to dedicated gamers, who are willing to invest hundreds or thousands of euros to be able to play games with maximum graphical settings. Cloud gaming services have become more common in recent years, but the bandwidth of current networks is not enough in most cases. Video games can be streamed from a remote server to any end device over the internet, thus making dedicated gaming consoles obsolete.

URLLC in 5G refers to the under one-millisecond latency. Latency is the time delay between the user's action and when the action happens. Low latency offers reliability which would allow 5G to be used in the medical field. Medical experts such as brain surgeons could do remote operations from another side of the world by using a robot that is designed for surgeries. This would allow surgeons to be available immediately in case of emergency. Working with a machine with near-zero latency would be the same as being at the operation yourself, except machines can have added functionality build into them, for example being able to detect and avoid unwanted scalpel cuts.

mMTC in 5G refers to the possibility of connecting up to million devices in a square kilometer area to a 5G network. This can be used to build smart cities where devices communicate with each other. One example of this could be fire detectors connected to the 5G network. Every fire detector in the city would be connected to a network and if a fire starts at any point, authorities will be informed and a firetruck would be automatically dispatched without requiring a person to see a fire and call the emergency number. [1]

3 Operations and Maintenance

Operations and Maintenance (O&M) is defined as a group of network management functions that provide network fault indication, performance information, and data and diagnosis functions [2]. BTS O&M team in Nokia MN develops and maintains 5G software components in BTS radio wave towers that monitor faults and performance statistics. The BTS configuration is also provided by BTS O&M.

Faults are monitored in Centralized Units (CU) and Distributed Units (DU) by a software component. If a fault is detected, an alarm will be raised and the network operators will be alarmed.

Performance statistics are also monitored in CU and DU by another of BTS O&M:s software components. If there are performance issues, an alarm will be raised similarly to the fault software component.

The test automation script that is theorized in this thesis is to be used for the software components that BTS O&M team develops and maintains.

4 Current implementation and problems

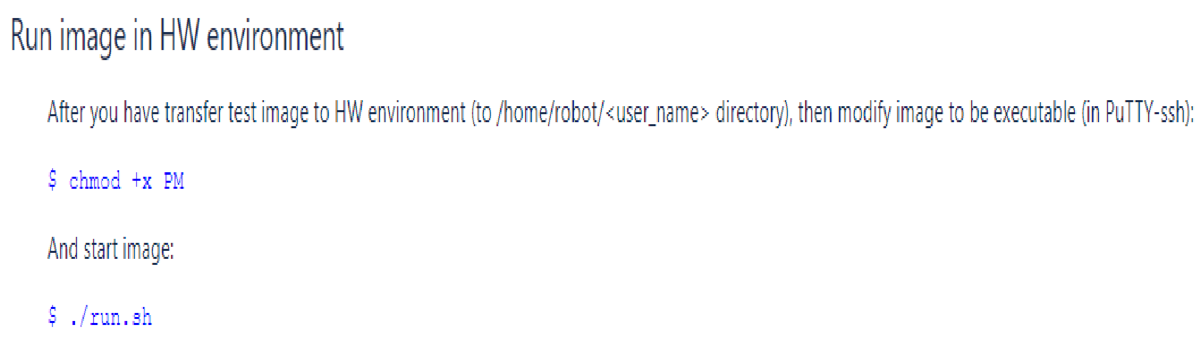
Currently, if a developer wants to test their code changes in a hardware test environment, they need to do multiple manual steps. These steps are documented in Nokia Mobile Networks (MN) internal documents [3][4].

The user must manually enable debugging logs and give permissions for the program to write logs to the terminal by modifying configuration files [3]. Sometimes the developer needs to manually update the environment to match the latest version or roll back the older version which is time-consuming and complicated [4].

In Nokia MN, it is a common way of working to do code changes in a cloud server, rather than having the program files locally in their work computers. Deployment of necessary software development environment and managing it is realistic only in Linux-based servers.

Currently, the software component’s code changes must be compiled and build in the development server and transferred to the test server by copying them from one to another. The most common way to work with remote servers is by mounting them as network drives in Windows File Explorer. A software developer will copy the build and packaged software component’s code binary from the development server to the test server. Windows uses File Transfer Protocol (FTP) for copying files between local drives or network locations.

Developers connect to both servers with a Secure Shell Protocol (SSH) connection. Login credentials are asked during login. See figure 2 below.



Run image in HW environment

After you have transfer test image to HW environment (to /home/robot/<user_name> directory), then modify image to be executable (in PuTTY-ssh):

```
$ chmod +x PM
```

And start image:

```
$ ./run.sh
```

Figure 2. Guide on how to give rights and start a binary of a software component. [3].

After the software components binary has been transferred to the test server, a developer must manually give executable rights to the binary and run it, as seen in the figure above.

The major problem with the current implementation is the need for manual configurations that developers need to take to get to test their code changes. This can be solved using automation. Automation is a way to save time, by making these configurations be done automatically with the use of an automation script.

5 Concept

The objective of this thesis is to theorize an automation script that makes everything possible with just a single command line input. The following paragraphs explain how the automation script should behave and what steps the software developer needs to take.

When a developer is done with their code changes for the software component they work on, they will push the code changes to Gerrit. After that the developer can connect straight to the virtualized hardware server, that has an automation script running. Either the user would have to run the script, or it is automatically run upon successful login to the server. This script asks for the developers Gerrit commit identifier and user credentials. The software component with the new changes will be fetched from Gerrit. Gerrit gets the software components code as a compiled binary from Jenkins.

The automation script will run all test cases on the software binary. Tests are fetched from online storage, rather than have them take space on the binary.

Logs from the tests will be saved to the file system and the user will get on-screen information written to the terminal on which tests were successful or failed.

The test automation would be implemented using Robot Framework. BTS O&M team members have used Robot Framework for the deployment of test automation on an older server. This script has not been ported to the current server. The Deployment chapter will be based on this script.

6 Virtualization

Virtualization is the use of software to create abstract layers on top of computer hardware. This chapter focuses on the virtualization of servers. Virtualization in this context means that the hardware capabilities are not directly accessed by the user but the hardware configuration is made into a virtual copy. Virtualization can be divided into machine virtualization or containerization [5][6]

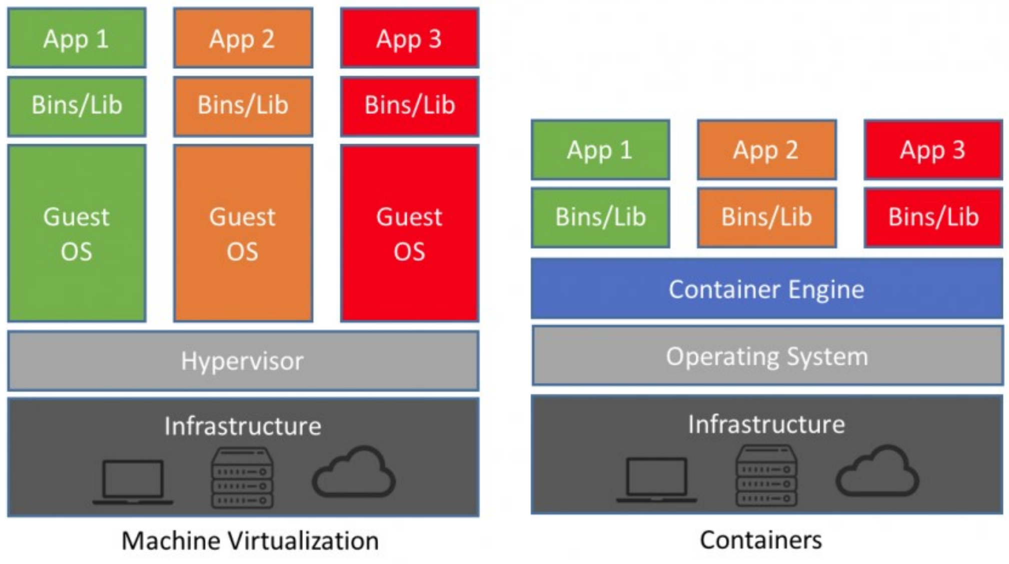


Figure 3 Machine virtualization and Containerization [6]

6.1 Virtual Machines

Machine virtualization is used to create Virtual Machines (VM). Figure 3 demonstrates the concept of machine virtualization. Physical server hardware is virtualized into multiple VM:s using a hypervisor. Virtualization turns the physical hardware of a server into a virtualized replica. This way a single server is divided into multiple smaller VM:s. Each VM has its own dedicated operating system, storage, and applications.

Hypervisor is a piece of software that enables physical hardware to be turned into a virtual replica of itself. This replica can be segmented into smaller virtual machines. Hypervisors can be divided into Type 1 and Type 2 hypervisors.

Type 1 hypervisors are installed directly on top of the physical hardware. Type 1 hypervisor is sometimes referred as a bare-metal hypervisor. Type 1 hypervisors are preferred, due to their direct access to hardware. Hypervisors that are directly connected to hardware are seen as the better performing and more secure hypervisors because there is no OS in between that adds communication latency. The lack of host OS makes the virtualization deployment more secure because having a host OS adds another vulnerability to the system.

Type 2 hypervisors are installed on top of the host OS. In this kind of implementation host OS is located between hardware and the hypervisor. Type 2 hypervisor is sometimes referred as a hosted hypervisor. Type 2 hypervisors cause more latency because communication between VM and hardware has to go through the host OS. The benefit of Type 2 hypervisors is that they are easier to setup and offer wider compatibility with different kinds of hardware configurations.

VM works like a traditional computer in the eyes of the end-user. Each VM is an independent unit, that is not reliant on the other VM:s. VM:s offer the benefit of having multiple operating systems installed on one server. One server can have for example its own VM for Windows, Debian, and Ubuntu. Moving VM:s between hypervisors is easy and flexible. These benefits come at the cost of resources required. If virtualization of a server is wanted where the operating system will stay the same in each VM, then the same OS will need to be duplicated in each VM, thus using lots of resources. Containerisation is the better option in these types of use-cases. [5][6]

6.2 Containers

Containerization is another way to approach virtualization. Figure 3 demonstrates the concept of containerization. Containerization is built on top of the OS, instead of hardware like VM:s. This limits every container in a containerized environment to have the same OS. The benefits of containerization are the lesser requirements for computing power and storage. Containerization avoids having to create multiple copies of OS across the containers, thus saving resources. The OS can be managed and configured for the whole container array, instead of having to manage and configure each OS like it is done with VM:s.

Containerization uses a container engine instead of hypervisors. A container engine is a piece of software that handles the user's communication between the hardware and the container. The container engine runs the containers from the end-users perspective.

Currently, Nokia MN uses containerization for their hardware test environments. vCU and vDU have been successfully deployed in VM:s, but it is beneficial to use containerization as there is no need for multiple operating systems. It is not realistic to have full 5G deployment for developers to test features. With containerization, a single cloud server can have multiple containers of Radio Access Network (RAN) units. Containerization is the logical choice for the task because the OS layer is the same in vRAN. [5][6]

6.3 vRAN

The hardware test server has an instance of virtualized Radio Access Network (vRAN) running as the environment. vRAN has multiple use cases, but in this instance, it can be seen as Radio Access Network (RAN) environment that operates exclusively in virtual space, without needing physical radio towers or connected end devices.

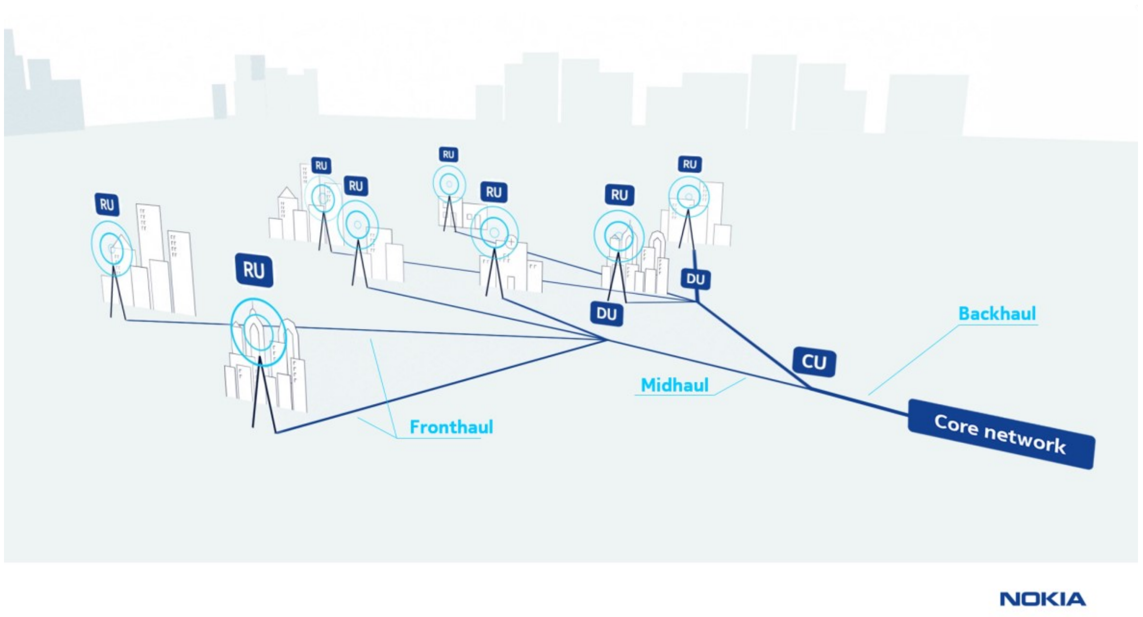


Figure 4. Radio Access Network visualized. [7].

vRAN consists of Core Network, Centralized Unit (CU), multiple Distributed units (DU), and multiple Radio Units. Figure 4 shows the topology of typical RAN deployment.

Core Network is the connection module from RAN to the worldwide internet. Phone operators use Core Network for routing calls and track data usage.

Centralized Units main purpose is to communicate and command distributed units. CU is also responsible for non-real-time network layer 2 and 3 functions.

Distributed Units answer to Centralized Units and communicate with Radio Units. DU is responsible for real-time network layer 1 and 2 scheduling functions.

Radio Unit is responsible for receiving, transmitting, and amplifying radio signals received from network devices, such as mobile phones. [7]

6.4 vCU & vDU

Virtualized hardware test server is a remote server that has an instance of virtualized Centralized Unit (vCU) or virtualized Distributed Unit (vDU) running. vCU and vDU mimic the real CU and DU that are in BTS radio wave towers.

The motive to test in these virtualized hardware servers is to be certain that the software component with the new code changes works in the actual CU or DU.

When making local changes in the development server, code will be tested by certain test frameworks. If changed code passes the tests in the development server then it can be expected that it will also work on actual BTS. However, it is advised to test in the virtualized hardware test server if the code interacts with other software components or if the memory allocation or timers are a major part of the code changes.

Programming languages, such as C++, can be used for memory manipulation. Sometimes developers need to allocate memory or use pointers that address the physical memory of the hardware the program is running on. Timings for RAN software components are crucial and missing them will cause issues for the whole component.

Nokia MN has separate test servers for vCU and vDU. Developer can reserve testing slot from an online tool for these servers. The servers have containers with an instance of vCU or vDU running that developers can use as their testing environment.

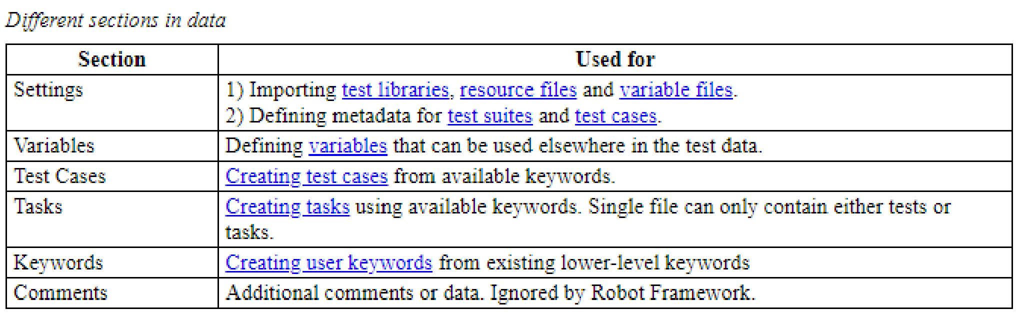
It is important to note that vCU and vDU communicate with each other. In vCU environment there exists a mimic vDU and the other way around. This mimic unit’s deployment is a simplified version whose purpose is to send responses to the main unit.

7 Robot Framework

Robot Framework is an open-source automation framework that is commonly used for test automation and robotic process automation. Robot Framework is open-source software, meaning anyone can use it for personal or commercial use cases [8].

Robot Framework was chosen for the task because it is widely used, thus having good documentation and because it has been developed originally by Nokia Networks. Robot Framework has been successfully used for test automation before in BTS O&M development team which was a major reason for choosing to continue with it as the automation framework. Robot Framework also supports third-party libraries that improve the potential of the tests.

Robot Framework is a keyword-driven test automation framework. This means that keywords act the same way as functions from traditional programming languages. This means that development is done by making smaller functions which makes the code easier to develop and understand.


Different sections in data

Section	Used for
Settings	1) Importing test libraries, resource files and variable files. 2) Defining metadata for test suites and test cases.
Variables	Defining variables that can be used elsewhere in the test data.
Test Cases	Creating test cases from available keywords.
Tasks	Creating tasks using available keywords. Single file can only contain either tests or tasks.
Keywords	Creating user keywords from existing lower-level keywords
Comments	Additional comments or data. Ignored by Robot Framework.

Figure 5 Robot Framework Sections. [9]

Data is defined in different sections as shown in the figure above. Different sections are recognized by their header row. The recommended header format is “*** Header ***”. [9]

Robot Framework uses tabular style syntax which makes writing test cases more user-friendly and readable. This means that each line of code made in Robot Framework uses a tabular separation of parts of the command.

Robot Framework supports custom libraries made in Python or Java programming languages. These libraries are a way to introduce new tools to expand the possibilities of testing. One of the most used third-party libraries is SeleniumLibrary which offers tools to interact with web browsers and -sites [10]. SeleniumLibrary has been used by BTS O&M in previous test automation systems.

Robot Framework programs are executed via a command-line interface, using Python. Python version 3 onwards is supported. The newest version of Python was downloaded and added to the system environmental variables.

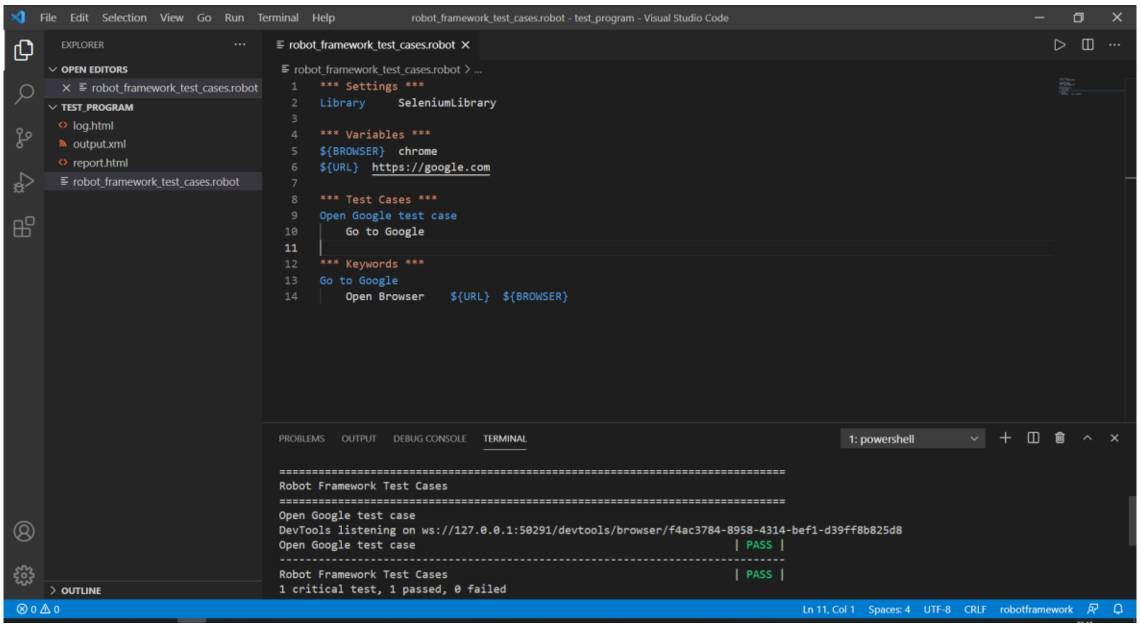
Robot Framework works in any text editor, but it is beneficial to use an Integrated Development Environment (IDE). IDE offers tools that make writing programs easier for developers, such as syntax highlighting, code completion, refactoring, and debugging.

The next task was choosing an IDE that fills the necessary requirements. The three main options for developing using Robot Framework are Robot Integrated Development Environment (RIDE), Visual Studio Code, and Eclipse.

RIDE is a development environment specifically made for Robot Framework, while Visual Studio Code and Eclipse require third-party extensions to support Robot Framework.

Visual Studio Code was chosen, because it offers a quality of life additions, such as easier working with other file types and integrated terminal for executing the tests. [11]

After downloading Visual Studio Code, Robot Framework Intellisense was downloaded from the Extensions marketplace. IntelliSense is a general term for various code editing features including code completion, parameter info, quick info, and member lists. IntelliSense features are sometimes called by other names such as "code completion", "content assist", and "code hinting”. [12][13]


File Edit Selection View Go Run Terminal Help
robot_framework_test_cases.robot - test_program - Visual Studio Code
EXPLORER
OPEN EDITORS
robot_framework_test_cases.robot
TEST_PROGRAM
log.html
output.xml
report.html
robot_framework_test_cases.robot
OUTLINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
Ln 11, Col 1 Spaces: 4 UTF-8 CRLF robotframework

```
*** Settings ***
Library     SeleniumLibrary

*** Variables ***
${BROWSER}  chrome
${URL}  https://google.com

*** Test Cases ***
Open Google test case
    Go to Google

*** Keywords ***
Go to Google
    Open Browser    ${URL}  ${BROWSER}
```

```
==============================================================================
Robot Framework Test Cases
==============================================================================
Open Google test case
DevTools listening on ws://127.0.0.1:50291/devtools/browser/f4ac3784-8958-4314-bef1-d39ff8b825d8
Open Google test case                                                 | PASS |
------------------------------------------------------------------------------
Robot Framework Test Cases                                            | PASS |
1 critical test, 1 passed, 0 failed
```

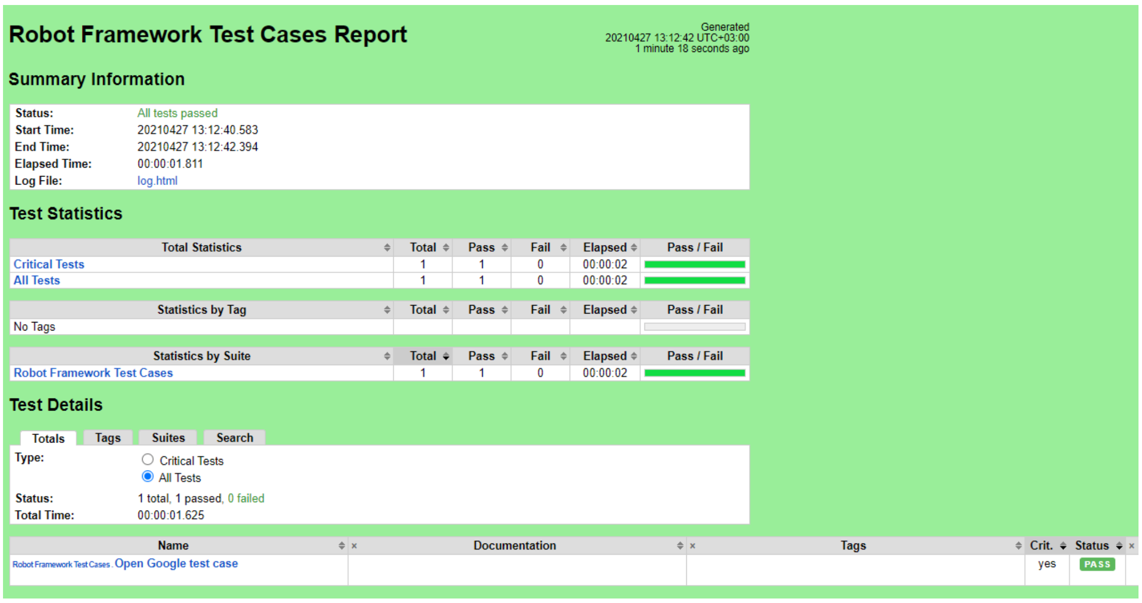
Figure 6 Robot Framework Test Program

The program seen in the figure above was developed to learn and demonstrate in-depth about how Robot Framework operates.

In the settings section, SeleniumLibrary is included. The variables section has two variables: ${BROWSER} and ${URL}. ${BROWSER} states which web browser is used and ${URL} states which URL to access. In the test cases section, we have a test case called “Open Google test case”. This test case calls “Go to Google” keyword. This user-created keyword calls “Open Browser” keyword which is defined in SeleniumLibrary. We give the two necessary variables for the call of the keyword which results in Google opening in Google Chrome browser. [10]

The test file is executed by inputting the command “python -m robot ./test.robot” to the integrated terminal in Visual Studio Code. This command states that “test.robot” file that is located in the current directory will be executed using python with robot module.

Upon completion of the tests, Hypertext Markup Language (HTML) files are generated. These files report the user about in-depth statistics of the tests. “log.html”, “report.html” and “output.xml” are the default artifact files that Robot Framework generates.



Robot Framework Test Cases Report

Generated
20210427 13:12:42 UTC+03:00
1 minute 18 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20210427 13:12:40.583
End Time:	20210427 13:12:42.394
Elapsed Time:	00:00:01.811
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:02	🟩
All Tests	1	1	0	00:00:02	🟩

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

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Robot Framework Test Cases	1	1	0	00:00:02	🟩

Test Details

Type: Critical Tests [✓ All Tests]
Status: 1 total, 1 passed, 0 failed
Total Time: 00:00:01.625

Name	Documentation	Tags	Crit.	Status
Robot Framework Test Cases . Open Google test case			yes	PASS

Figure 7 report.html

The figure above shows the details in report.html. We can see the details of the test completion such as which tests passed when the tests were run and how long it took to complete the run.

8 Gerrit & Jenkins

Gerrit is a web-based code collaboration tool designed for software developers. Gerrit works as a storage space for code changes where they get reviewed and tested before being allowed to submit to the production branch. In the development server, git commands are configured to automatically push to Gerrit. Once developer pushes their code changes using git commit and git push, Gerrit will automatically trigger Jenkins. [14]

Jenkins is an open-source automation server. A popular use case for Jenkins is to automate software testing in the conjuration of code collaboration tools, such as Gerrit. Jenkins will compile the software component to a binary. After it has been compiled, Jenkins will proceed to run unit and system tests for the software components binary. The developer can follow the process from the Jenkins pipeline which visualizes the testing process. [15]

The code changes can be merged to the production branch when it has passed Jenkins verification and two other developers have given their permission as a +1 and +2 mark. These marks mean that the changes are implemented according to commonly agreed coding style guidelines and fulfill the specification required for the feature.

Gerrit would be used to replace the manual copying of build files between the development server and hardware test server. Once the user connects to the hardware test server, the automation script would ask for the developer's Gerrit credentials and they would be able to choose the change to commit from a list of all their previous code reviews. When a developer chooses their code change, the build files are fetched from the linked Jenkins page and the testing process would begin.

9 Previous implementation

This chapter will analyze the previous test automation system that has been in use on older virtualized hardware test servers. The automation system has been developed by other members in BTS O&M. The chapter will analyze the test automation part made using Robot Framework and explain the deployment steps necessary to get the test automation working. The system has not been ported over for use in the current virtualized hardware test servers. The deployment steps for the automation system that is theorized in this thesis are identical to the previous implementation.

9.1 Deployment steps

This automation system consists of scripts made in multiple scripting languages, where the combination forms a comprehensive automation deployment.

The automation system starts by connecting to the environment and artifactory. Artifactory is a storage space for by-products of the software, such as log files and snapshots of the state of the program. The automation system will push and pull data between the test server and artifactory during the run, such as log files and snapshots. After making necessary connections, the automation system will move to the setup stage.

The setup stage consists of fetching build files from the artifactory to the environment. Once the files have been fetched, they will be unpackaged to the file system of the environment.

The next step is to modify build files according to environment configurations. After the modifications have been done, the system will check if there are builds from previous runs. The old builds will be deleted and the current build shall be deployed in its place.

Robot Framework is used for the deployment steps from commissioning onwards. After commissioning has been done, the system will check if the software component binary that has code changes is running and will have tests run on it.

The user will get real-time status updates in the terminal. If something fails during the deployment at any point, the user will get error notifications and the current build will get stopped and cleared from the environment.

Logs and snapshots will be saved to the file system of the test server and in artifactory. Log files are used by the developer to determine what went wrong with the deployment. Log files have timestamps and information that was also outputted to the terminal.

Teardown of the environment will be executed at the end of the deployment process which resets the environment to the default state.

9.2 Robot test suites

Robot Framework tests are used to execute the remaining deployment steps. The tests made with Robot Framework in the automation system have been divided into three files. A set of tests in the same file is called a test suite. Each test suite represents a remaining deployment step. The remaining deployment steps are commissioning, testing, and ending the deployment.

The execution order for the test suites has been listed in a text file. The system will look for the text file during the build and pass it to the robot configuration, where it is used to define the execution order.

The first test suite is in charge of commissioning. The tests suite includes two test cases. The first test case's purpose is to do the commissioning. The test will call a keyword that opens a web browser and logs into a remote server. The automation for opening the web browser and inputting values to the login fields is done using SeleniumLibrary [10]. Once the connection has been established, commissioning will be completed with the remote server. The secondary test case's purpose is to take screenshots of various elements during the commissioning in the remote server.

The next test suite is executed after the first one is done with the commissioning. The test suite runs the test cases in the virtualized hardware environment for the software components’ binary. This will check that the software binary can be run successfully and runs test cases on it. The results will be printed on the terminal and collected as log files.

The final test suite contains a test case that handles the ending of the deployment process. This includes taking screenshots, handling logs and running a teardown of the environment. Logs are saved to the file system of the server and to the artifactory. This test suite also runs a teardown of the environment, which resets the environment to the state it was before the deployment. After this test suite is executed successfully, deployment is complete.

10 Conclusion and future steps

This Bachelor's thesis aimed to find out if automation of virtualized hardware test environment were plausible and to learn more about how such environment operates and what backend resources and systems would be needed.

The objective was accomplished by studying virtualization, vRAN, Robot Framework, Gerrit, Jenkins, and the old implementation of test automation. The study was completed successfully. Based on this study and the previous experiences of the development team, it can be concluded that automation for virtualized hardware environments is plausible and beneficial.

This Bachelor’s thesis can be used in the future as a reference if BTS O&M wants to implement the automation script that has been conceptualized in this thesis. Making the actual implementation of such an automation solution is a complex task that requires lots of technical understanding, resources, and time allocation.

References

1 Part 1: The truths and myths of 5G deployment – a technical perspective. 18 October 2019. Online. NGP Capital. <https://ngpcap.com/news/5g-impact-1> Accessed May 10, 2021

2 Guidelines for the Use of the "OAM" Acronym in the IETF. June 2011. Online. Internet Engineering Task Force (IETF). <https://datatracker.ietf.org/doc/html/rfc6291> Accessed May 13, 2021

3 How to test in real HW. Online. Nokia Networks internal document. Accessed March 10, 2021

4 How to update CU environment. Online. Nokia Networks internal document. Accessed March 10, 2021

5 Virtualization. 19 June 2019. Online. IBM. <https://www.ibm.com/cloud/learn/virtualization-a-complete-guide> Accessed April 30, 2021

6 Containers vs. Virtual Machines (VMs): What's the Difference?. 16 March 2018. Online. NetApp. <https://blog.netapp.com/blogs/containers-vs-vms//> Accessed April 30, 2021

7 Open RAN Explained. 16 October 2020. Online. Nokia. <https://www.nokia.com/about-us/newsroom/articles/open-ran-explained/ Accessed April 30, 2021

8 Robot Framework. Online. <https://robotframework.org> Accessed March 10, 2021.

9 Robot Framework User Guide. Version 4.0.1. Online. < https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html> Accessed March 11, 2021

10 SeleniumLibrary. Github. Online < https://github.com/robotframework/SeleniumLibrary> Accessed April 2, 2021

11 Visual Studio Code. Online. <https://code.visualstudio.com/> Accessed March 11, 2021

12 Visual Studio Intellisense. 31 March 2021. Online. Visual Studio Code. < https://code.visualstudio.com/docs/editor/intellisense> Accessed April 22, 2021

13 Robot Framework Intellisense. Visual Studio Marketplace. Online < https://marketplace.visualstudio.com/items?itemName=TomiTurtiainen.rf-intellisense> Accessed April 22, 2021

14 Gerrit. Online. <https://www.gerritcodereview.com/> Accessed May 2, 2021

15 Jenkins. Online. <https://www.jenkins.io/> Accessed May 2, 2021