Expertise
and insight
for the future

Jesse Nygrén

Automating Virtual Test Environment Creation on VMware vSphere Platform

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

26 January 2020

Author	Jesse Nygrén
Title	Automating Virtual Test Environment Creation on VMware vSphere Platform
Number of Pages	35 pages
Date	26 January 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Antti Piironen, Principal Lecturer

The aim of this thesis was to create a solution for Airbus Defence and Space Oy to automate virtual test environment creation and management processes. The solution had to be compatible with other technologies used by the company and it had to be created on the VMware vSphere platform.

The software that was developed in the project was written in the Python 2.7 programming language. VMware vSphere Automation API requests were used to automate virtual environment creation on vSphere platform.

The final product was able to generate any size on virtual environments for multiple purposes. On top of the virtual environment creation process, the software can communicate with the virtual environment and generate configuration files for Airbus Defence and Space's software.

The phases of the developing process were planning, software development, testing, and implementation. The process started with planning and defining the project with stakeholders. The software development phase was done independently while consulting the Radio Console System (RCS) team leader. After the software was developed, it was tested during the testing phase. The testing included testing the virtual environment. The goal with the virtual environment test was to find out best practices and parameters for virtual machine deployment and power on functions to maximize efficiency. After the tests were done, the software was implemented to be a tool for integration and verification teams.

The developed solution turned out to be a scalable, working, and efficient tool for integration and verification teams who are responsible for testing the software of Airbus Defence and Space. The automation of the virtual environment creation with the software saved time, workhours, and computational resources.

Keywords	VMware, REST, API, Python, virtualization, automation

Tekijä Otsikko Sivumäärä Aika	Jesse Nygrén Virtuaalisten testiympäristöjen luonnin automatisointi VMware vSphere-alustalla 35 sivua 26.1.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Yliopettaja Antti Piironen

Insinöörityön aiheena oli virtuaalisten testiympäristöjen luonnin automatisointi vSphere-alustalle. Tavoitteena oli kehittää Airbus Defence and Space Oy:lle järjestelmä, joka automatisoi virtuaalisten testiympäristöjen luonnin ja hallitsemisen. Järjestelmän tuli olla yhteensopiva yrityksen muiden käyttämien teknologioiden kanssa, ja se tuli toteuttaa VMwaren vSphere-alustalle. Työnä toteutettiin Python 2.7 -ohjelmointikielellä kehitetty ohjelma, joka automatisoi virtuaalisten testiympäristöjen luonnin VMware:n vSphere Automation API:a käyttämällä.

Lopullinen tuote kykeni generoimaan rajoittaman kokoisia virtuaalisia ympäristöjä useisiin erilaisiin tarpeisiin. Virtuaaliympäristöjen luonnin lisäksi ohjelma kykeni kommunikoimaan luodun virtuaaliympäristön kanssa SSH-yhteydellä ja generoimaan konfiguraatiotiedostoja Airbus Defence and Spacen ohjelmistoille.

Ohjelmistotuotantoprosessin vaiheita olivat suunnittelu, ohjelmistokehitys, testaus ja käyttöönotto. Kehitystyö alkoi suunnittelusta, jota tehtiin yhdessä sidosryhmien kanssa. Ohjelman kehitysvaihe toteutettiin itsenäisesti samalla konsultoiden RCS-tiimin tiiminvetäjää. Kehityksen jälkeen ohjelmistoa ja sen luomia virtuaaliympäristöjä testattiin. Virtuaaliympäristöjen testauksessa selvitettiin muun muassa optimaalisia ratkaisuja ja parametrejä virtuaalikoneiden luonnissa ja käynnistämisessä ajankäytön tehostamiseksi. Testien ja korjausten jälkeen ohjelma otettiin käyttöön ja tarvittavat virtuaaliympäristöt luotiin.

Kehitetty ratkaisu osoittautui skaalautuvaksi, toimivaksi ja tehokkaaksi työkaluksi integraatio- ja verifikaatiotiimeille, jotka vastaavat muun muassa ohjelmistotestauksesta. Virtuaalisen testiympäristön luonnin automatisointi ohjelmalla loi säästöjä työtunneissa, laskentaresursseissa ja ajassa.

Avainsanat	VMware, REST, API, Python, virtualisaatio, automaatio

Contents

List of Abbreviations

API	Application programming interface. An interface for cross communication between two programs.
CPU	Central processing unit. It is the main processor of a computer.
CRUD	Create, read, update, and delete. Four major database functions.
CSV	Comma-separated values. It is a text file format where values are separated with commas.
DHCP	Dynamic Host Configuration Protocol. A network protocol by which IP addresses are dynamically assigned to client devices.
GUI	Graphical user interface. A user interface that includes graphical elements such as windows, buttons, and icons.
HTTP	Hypertext Transfer Protocol. It is a network protocol defining message format and transmission used by the World Wide Web.
IP	Internet Protocol address. It is the numerical address of a device using computer networks.
MAC	Media Access Control address. It is the unique identifier of network interface controller.
OVF	Open Virtualization Format. It is a standard for packaging software to be run on virtual machines.
RCS	Radio Console System. RCS 9500 is TETRA dispatching console software of Airbus Defence and Space.
REST	Representational state transfer. A software architecture used in Web service development.

TCP	Transmission Control Protocol. It is a standard for network communication establishment.
TETRA	Terrestrial Trunked Radio. It is a standard for trunked radio systems and professional mobile radios. The main users of TETRA are government and public safety officials.
UUID	Universally Unique Identifier. It is a 128-bit number identifier for information identification in computer systems.
VM	Virtual machine. It is an emulated computer system run on a hypervisor.

1 Introduction

Virtualization can be considered as one of the most important technologies in the modern IT industry. It helps companies to cut down spending resources, such as time, energy, money, and space, while being a more environmentally sustainable option compared to traditional non-virtualized systems. [1; 2.]

The thesis was commissioned by Airbus Defence and Space Oy. This company develops and researches secure land communication solutions and devices for customers working on public safety, military, healthcare, utility, and transportation fields. The company has two sites in Finland, in Helsinki and Jyväskylä. [3.] The software project documented in this thesis was developed on the Helsinki site of Airbus Defence and Space.

There was a need to develop a solution that was able to generate virtual test environments for testing capacity related features of Airbus Defence and Space's RCS 9500 dispatcher solution. The need for this kind of solution was introduced by the RCS team leader, who works in the integration and verification department. The need for a virtual environment was real since the test that was about to be conducted demanded 255 individual computers. If the same environment had been built on actual hardware, the costs would have been extremely higher.

The goal of the final year project was to develop a solution which can automate virtual test environment creation, configuration, deletion, and management on the vSphere platform of VMware. The key requirements for the system were to be scalable, meaning it had to be able to generate different sizes of virtual environments, and to be compatible with the other technologies used by the company. The software of the project was developed with the Python 2.7 programming language and it was built around VMware vSphere Automation API which is used to automate processes in the vSphere environment.

2 Key Concepts

2.1 Virtualization

The term virtualization means a virtual creation of something actual. In practice, it means creating virtual versions of operating systems, servers, storages, or networks. The virtualized versions of the actual counterparts are created with software that can emulate the functionality of the hardware. [1.]

The use of virtualization is growing, but it is not a new invention. The development of virtualization technology started in 1960 when IBM created a computer system, IBM S/360-40, which was able to perform full virtualization. IBM innovated the virtualization technology and began the active development work, soon after other companies joined in. From 1960 to 1998 companies such as General Electric Company, Burroughs Corporation, AT&T, Microsoft, Locus Computing Company, and Connectix were the companies developing virtualization technology. In 1998 a new company called VMware was founded. In 1999 they released the first ever virtualization product capable of virtualizing the x86 system, the VMware Workstation. Since then VMware has been one of the leading developers of virtualization technology. [4, p. 6-8.]

In traditional architecture the operating system is installed on the hardware and needed applications are installed on the operating system. In virtual architecture the process is different. In a virtualized server the virtualization software layer, called a hypervisor, emulates the needed computer hardware such as central processing unit (CPU), memory, network, and storage resources. The hypervisor detaches physical resources from the actual environment and divides resources as they are needed for the virtual environment. The virtual machines of the virtual environment, which are basically emulated computers, think they are using real hardware while they are actually using the emulated hardware of the hypervisor. [1; 5.] The hypervisor is responsible for actively sharing hardware resources for virtual machines, as computational demands change, to maintain fluent operation of the system [1]. It also enables virtual networks and clustering abilities for the virtual environment [6]. The comparison of a traditional and virtual environment setup is displayed in figure 1.

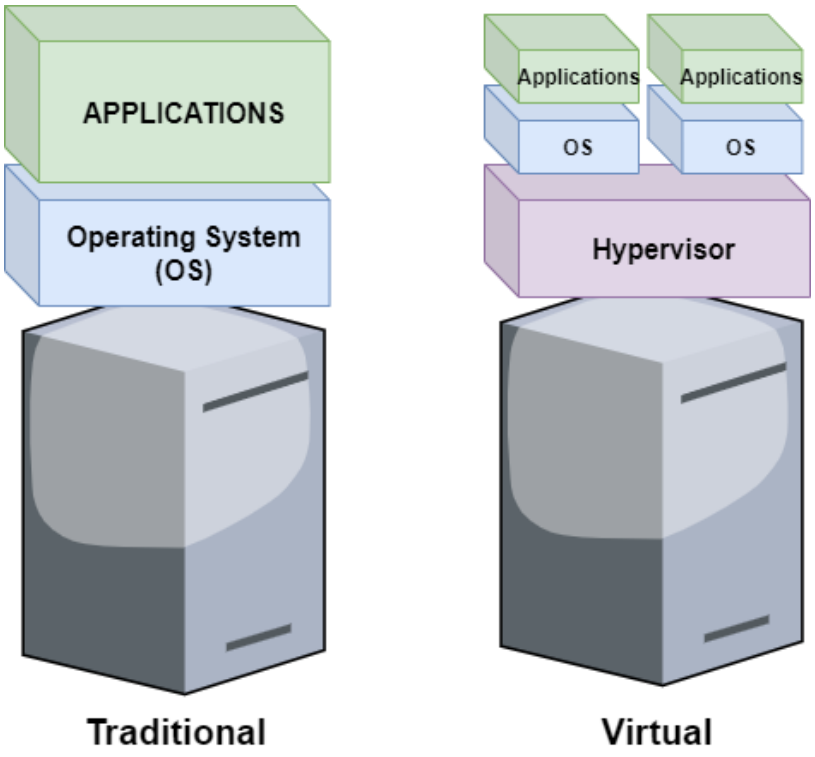

Figure 1. Comparison of traditional and virtual architecture [1]

Virtualized environments have many advantages and benefits over traditional environments. Virtual environments are a commonly cheaper and more environmentally friendly option for companies than traditional environments. Traditional environments need more physical hardware and space while virtual environments can achieve the same productivity on less hardware, thus leading to lower space requirements, lower costs, and smaller carbon footprint. Virtual environments are also easier and more time efficient to administrate, maintain, setup, and tear down. They are also easy to backup, recover and move to a completely new location if needed because they run on less hardware. [1.]

Virtualization was selected as the core technology for the final year project since it is a superior approach when comparing traditional resource heavy systems. The program which was developed was built around the VMware vSphere virtualization platform.

2.2 REST API

REST, representational state transfer, is an architectural design enabling cross platform communication. It is built on the Hypertext Transfer Protocol (HTTP) and is especially used in applications that are using the HTTP protocol over the Internet for communication between the client and the server. It was invented in 2000 by Roy Fielding who is also one of the developers of the HTTP standard. [7; 8.]

REST is all about resources and handling them. In REST, resources can be basically anything from documents to images to non-virtual objects and so on. Resources are the part of the URL which are separated with the slash character after the domain address. For example, an image resource could be presented in the URL form as http://final-yearproject.api/image where /image is the image resource. The resource handling can be done with the GET, POST, DELETE, and UPDATE HTTP methods which are equivalent to the create, read, update, and delete (CRUD) methods in database handling. [7; 8.]

The GET method can be used in the same way as the read method of CRUD. If the user wants to return the image resource from http://finalyearproject.api/, it could be done for example with a “http GET http://finalyearproject.api/image” request. The POST method is equivalent to the create method of CRUD. The information that is to be created can be for example in the XML, eXtensible Markup Language, or JSON, JavaScript Object Notation, format. The request could look like this: “http POST http://finalyearproject.api/image”. The counterparts for DELETE and UPDATE are also included in the CRUD methods. [7.]

The VMware vSphere Automation API which is a REST API was used in the final year project. With the REST API it was possible to automate test environment creation processes and thus save resources such as time and money.

2.3 SSH

Secure Shell, which is often abbreviated as SSH, is a protocol for secure communication through a text interface between a client computer and a server. The technology was invented in 1995 by a Finnish developer, Tatu Ylönen. Ylönen developed the protocol as there was a demand for a secure way to remotely login to servers over the Internet after a hacking incident had occurred in a Finnish university network. The first version of the SSH protocol was called OpenSSH and it was an open source product. [9.]

Secure communication between the server and the client is secured with symmetric encryption, asymmetric encryption, and hashing [10]. The SSH connection can be divided into three steps.

In the first step the server listens to a specific port for an SSH connection. The default port is 22. The client initiates a Transmission Control Protocol (TCP) handshake with the server. [11.]

In the second step, by using the Diffie-Hellman algorithm both parties create public keys and private keys with the same seed number. A shared secret key is created from the generated public and private keys to secure the connection. [11.]

In the third step the server sends encrypted information to the client. The encryption is done with the public key. The client can decrypt the information from the server by using the private key. [11.]

SSH technology was used in the final year project to communicate with the virtual machines. The vSphere Automation API lacked the ability to send commands remotely which made the SSH technology the only viable means of communication.

3 Overview of Used Products

3.1 VMware

VMware is a company specializing in virtualization and cloud computing products. The company was founded in February 1998 in Palo Alto, California. Their first product, Workstation 1.0, was introduced in 1999. From 1998 to this day the company has grown from a small firm employing only 5 people into a massive corporation of over 20,000 employees and half a million customers. [12.]

VMware offers a wide selection of virtualization solutions for both private and company customers. They focus mainly on virtualization platforms and virtual machine administration, but they also offer products for IoT, cybersecurity, and network administration needs. Their vSphere virtualization platform is the industry-leader in virtualization products. [13.]

3.1.1 VMware vSphere

VMware vSphere is not a single piece of software but a complete virtualization platform containing many products from VMware that are needed to create a fully virtualized system. The core products are VMware ESXi, VMware vCenter Server, and VMware vSphere Client. The newest version of the VMware vSphere is version 6.7 and it was used in the final year project. [14; 15.] The topology of the VMware vSphere system is displayed in figure 2.

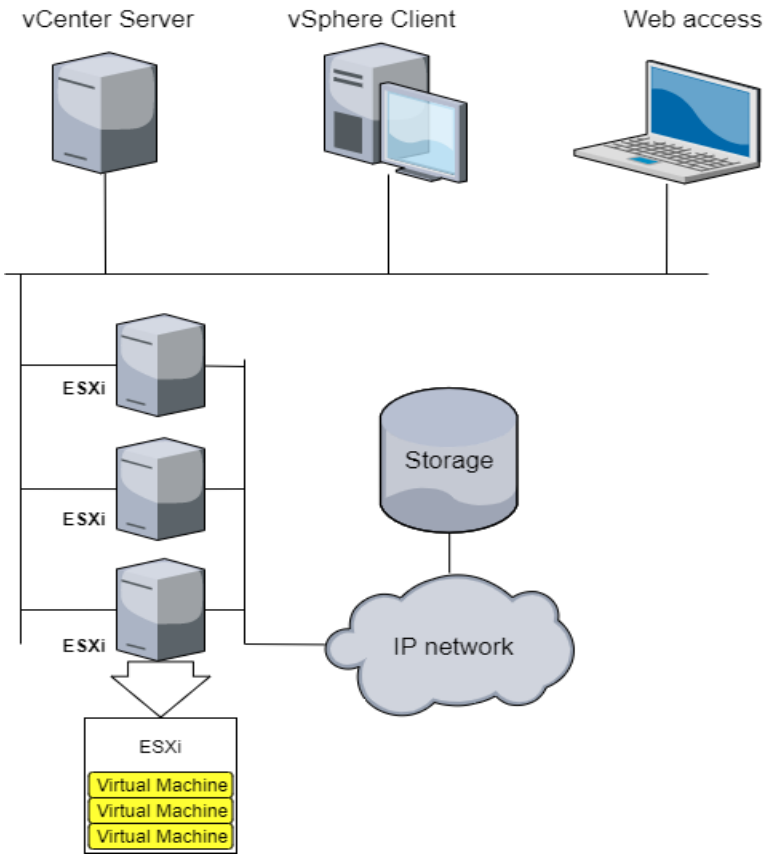

Figure 2. VMware vSphere topology [16, p. 10]

VMware ESXi, or Elastic Sky X Integrated, is the hypervisor included in the VMware vSphere suite. It is a type-1 hypervisor that runs directly on system hardware without an operating system. This kind of hypervisors are also called bare-metal hypervisors. VMware ESXi is lightweight, only 150 MB, and it packs several useful features for enterprises such as memory ballooning, traffic shaping, logging, a graphical user interface (GUI), and vSphere PowerCLI. [17.] The virtual machines of VMware vSphere are created on VMware ESXi.

VMware vCenter Server is the administration server of VMware vSphere that runs on a Suse Linux based Photon OS operating system. It allows system administrators to manage ESXi hosts and virtualized appliances, such as virtual machines, networks, and data storages, from a centralized location. vCenter Server can be accessed through an HTML5-based vSphere web client or the vSphere C# client. The C# client is depreciated from version 6.7 onwards and the use of the web client is recommended. The latest

release of VMware vCenter Server, version 6.7, can handle the maximum amount of 2,000 ESXi hosts and 35,000 virtual machines. [18.]

3.1.2 vSphere Automation API

VMware offers vSphere Automation REST API to control and automate the vSphere platform. It is an HTTP-based REST API and uses the standard HTTP methods: GET, POST, PUT, PATCH, and DELETE. The request URIs are HTTPS requests. The vSphere Automation REST API is divided into five different sections: *cis*, *appliance*, *content*, *vapi,* and *vcenter*. [19.]

The *cis* API contains calls to handle session management. With the *cis* API users can for example authenticate themselves and create new sessions which are required to use and operate the vSphere platform. [19.] Session management calls from the *cis* API were used in the final year project.

The *appliance* API contains calls to administrate different appliances that are part of the system. It offers many different API calls that are very useful for vSphere system administrators such as calls to perform network and system management, health management, and remote access with different connection types. [16.] The *appliance* API was not used in the final year project.

The *content* API contains calls for global configuration settings and library management in the Content Library Service. With this API the user can for example configure the Content Library Service and perform CRUD actions and file management in content libraries. [19.] Content libraries can be used to store different kinds of files on the vSphere platform and they were used to store virtual machine templates in this project. Content library CRUD calls from the *content* API were used in the final year project.

The *vapi* API contains calls for metadata authentication [19]. The *vapi* API did not contain any relevant API calls to be used in the final year project.

The *vcenter* API was the most used vSphere Automation API in the project. It provides API calls for managing the VMware vSphere environments. These calls include operations such as virtual machine creation, virtual machine deployment from a template, powering on and off virtual machines, and getting the state of a virtual machine. [19.]

3.2 Python 2.7

Python 2.7 version was released in July 2010 and it is the last version of the 2.x series. The support for Python 2.7 ended on December 1, 2020, meaning it does not get any new bug fixes anymore. [20.]

The Python version used in the project was Python 2.7. It was selected due to system compatibility with the system used by the test automation team.

3.3 Windows Server 2019 Server Core

Windows Server 2019 Server Core is the most minimalistic version of the Windows Server 2019, meaning it requires less hard disk space compared to the regular version. The main difference compared to the regular version is that the Server Core version does not have a GUI. The Server Core is controlled using the Windows command line or Windows PowerShell. The Server Core also lacks audio support, accessibility tools, and out-of-box experience features, but they were not required in the use cases of the final year project. [21.]

The lightness and sufficient set of features of the Windows Server 2019 Server Core led to the decision to use it in the project. The operating system turned out to be an excellent selection for large scale virtual environments.

3.4 OpenSSH

OpenSSH is an open source version of SSH maintained and developed by developers from OpenBSD Project [22; 23]. It is available on many different platforms that are mainly

UNIX-based. The latest addition in supported platforms is Microsoft Windows products [24]. OpenSSH has been included in Windows 10 and the desktop version of Windows Server 2019 since autumn 2018 [23].

The OpenSSH version used in the project was release 8.0.0.0. It had to be manually installed since it was not included in the Windows Server 2019 Server Core version by default.

4 Software Planning

The final year project started by planning and defining required features and functionalities. The software that was to be developed had to be defined properly, the technologies which were going to be used had to be selected, and the needed hardware resources had to be requested from the manager of the laboratory team.

The project initially started from a need for a virtual platform to test the connectivity of hundreds of RCS 9500 Terrestrial Trunked Radio (TETRA) dispatchers with a TETRA DXT. The project got a green light from the managers after the benefits and resource saving possibilities were introduced. The system that was about to be built would save tremendous amount of manual work, hardware resources, time, and money.

The planning of the project involved many stakeholders. The project was defined with the product owner who works in the integration and verification department as a team leader for testing the RCS 9500 solution. The needed hardware resources for the project were organized, managed, and administrated by the laboratory team, their system administrators, and the manager. The manager of the laboratory team was the person in charge for ordering the needed hardware while the team members assembled the hardware and were responsible for the administration and maintenance of the equipment.

4.1 Selecting the Platform

The technologies used in the project were selected with the product owner. The main requirements for the technologies were easy maintainability, cost efficiency, and compatibility with other products of Airbus Defence and Space.

The initial plan was to create a program that was able to generate a fully configured test environment on the Docker platform, but the plan was cancelled. Using the Docker platform was an extremely good idea for it would have been the most efficient way of executing the needed tests. However, it was impossible to use the platform due to driver restrictions.

The platform that was selected was the VMware vSphere platform. The fact that the company already had a solid and broad VMware infrastructure and competent people behind it led to the decision to use it. The VMware approach was not the first option to select since it is not as lightweight as Docker. VMware creates a full instance of an operating system for each virtual machine while Docker containers are managed by the kernel of the host system, thus saving resources. [25.]

4.2 Hardware Resources

Hardware resources were planned by the product owner in cooperation with the manager of the laboratory team to meet the requirements to run a software test on 255 individual virtual machines. The hardware platform used in this project was a high availability cluster designed by a test systems engineer from Airbus Defence and Space who carried out a study about commissioning a high availability VMware cluster [26].

Each server on the cluster hosted an instance of an VMware ESXi hypervisor. The whole capacity of the cluster was not used in this project, but a separate test resource pool was created. The created resource pool reserved a part of the resources of the cluster, such as computational power and memory, to be used only inside the resource pool. This kind of procedure secures that other services running on the same cluster do not interfere with the services running in the resource pool. Detailed information about the cluster and the test resource pool can be found in table 1.

Table 1. Hardware specifications

VMware vSphere	
ESXi version	6.7.0 Update 3
vCenter version	6.7.0 Update 4
HA (High Availability) Cluster	
Server amount	4
CPU Cores	16
RAM	256 GB
Test Resource Pool	
CPU speed	104 GHz
RAM capacity	84 GB
Network	1 Gb/s

VMware DRS was enabled to ensure balanced load distribution inside the cluster. DRS, or Distributed Resource Scheduler, is a technology that automates load distribution inside the cluster for optimized performance. When a virtual machine is powered on, the DRS assigns it to an appropriate ESXi host from the cluster. The automated selection is based on the overall load balance of the cluster. [27.] A dynamic host configuration protocol (DHCP) server was configured for the resource pool by the laboratory team to automatically assign Internet protocol (IP) addresses to all deployed virtual machines when they are powered on.

4.3 Defining and Planning the Software

The definitions to the project software were set together with the product owner. The definitions were done to prevent the software development to not to go on side rails but to maintain the focus on important matters.

The programming language for the software was agreed to be Python 2.7. The language was selected due its compatibility with tools used by the test automation team. The software was defined to be built on the vSphere Automation API.

Since the vSphere Automation API is a reasonably large collection of different API calls with different functionality, it was necessary to select only the ones needed for the project. The functionalities needed were virtual machine deployment from a template, the ability to get virtual machine information such as Universally Unique Identifier (UUID) and IP address, the ability to delete created virtual machines and the ability to power deployed virtual machines on and off. The needed API calls were selected for use from the vSphere Automation API.

On top of the VMware functions, the software needed also to remotely interact with the newly created test environment of virtual machines. The need for a way to remotely upload and download files and send scripts and commands to the virtual machines was defined.

The main usage of the software that was going to be developed was to be a tool for integration and verification teams. The software also needed to generate all configuration files that were going to be used to initialize the test cases of test automation.

After the scope for the software was defined, it was time to plan the architecture. The architecture was designed by the author of this thesis, Jesse Nygrén. The initial plan was to create an architecture that makes the future development and use of the code as flexible as possible. A sequence diagram displaying the functions of the software is shown in figure 3.

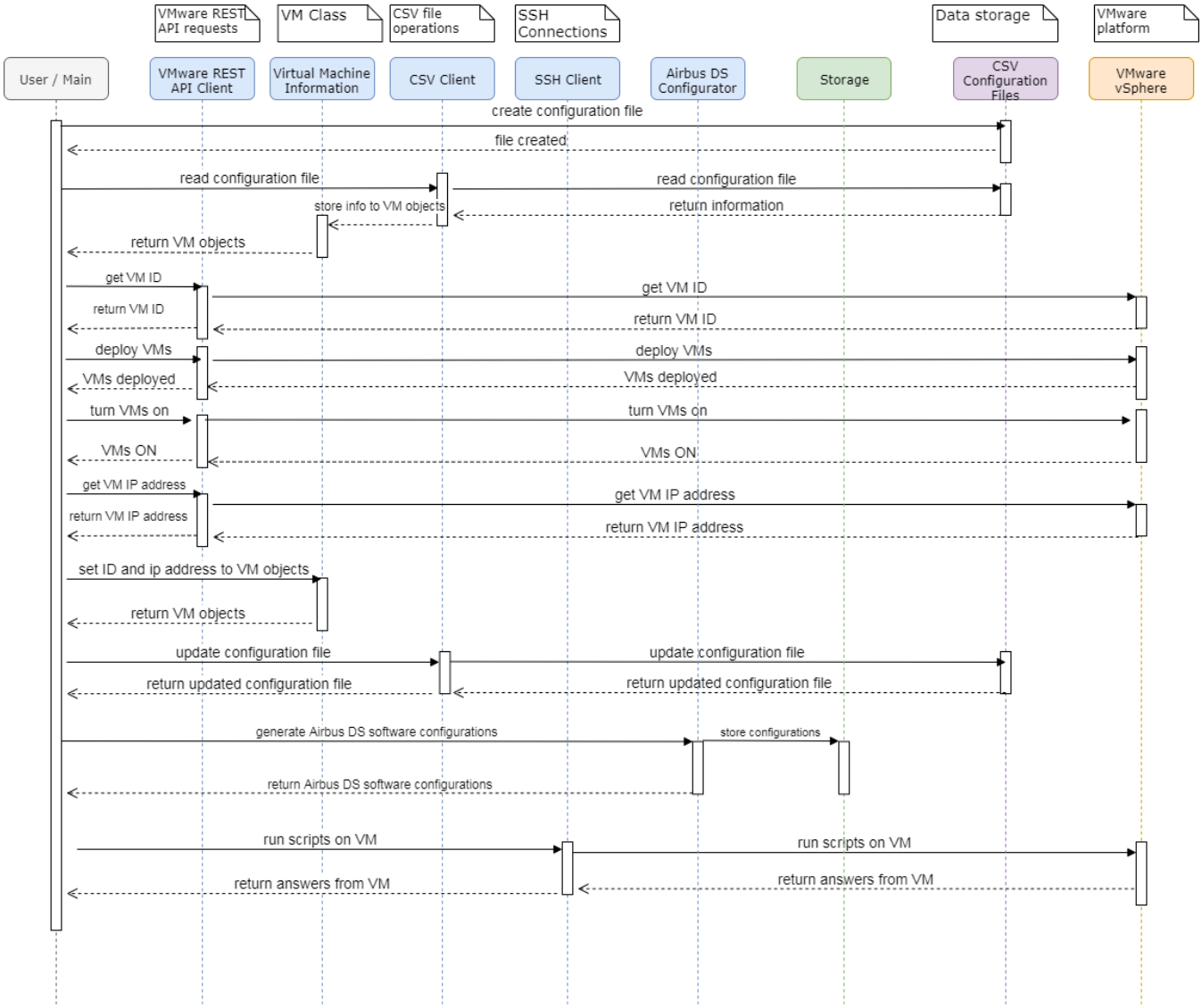

Figure 3. The sequence diagram of the software

The architectural design of the software is a library oriented architecture. In a library oriented architecture, each Python module works as an independent and separated library and can be used independently or in parallel with other libraries. [28]

4.4 Creating a Ready-to-Deploy Template

The vSphere has two different options to make clones from virtual machines: the cloning function and the template deployment function. The cloning function lets the user to clone new virtual machines from existing virtual machines that can be either powered on or off. The cloned machines will have newly generated Media Access Control (MAC) addresses

and UUIDs so they can be treated as completely new instances rather than pure clones. The clones will have all the same installed programs and configurations as their parent virtual machine has. [29; 30.] The vSphere Automation API does not support the cloning feature. [19]

Because of the lack of a cloning feature in VMware REST API, the options were to create new virtual machines from scratch or to deploy new virtual machines from templates. The deployment from templates was chosen to be the approach due to the ease of use.

Deploying virtual machines from templates using the vSphere Automation API resembles the cloning function with some differences. The user needs to create a virtual machine, install the needed programs and configurations, and then create a new virtual machine OVF template from the created and configured virtual machine. The template will be a static master clone from the target virtual machine and it cannot be altered after the creation. If the user wants to make changes to the template, the user has to make the changes to the target virtual machine and deploy a new template from it. To be able to deploy the template using the vSphere Automation API, the template needs to be placed in a content library from where the API call handling OVF template deployments can use it. [19; 30.]

The clone function would have been a more time efficient approach compared to the template deployment function. The clone function does not require a separate template from where it can make the clone and it can create the clone in the same location where the parent virtual machine is located, thus not requiring to get it from a separate content library.

The virtual machine template that was going to be used in the project was built and configured on Windows Server 2019 Server Core operating system. Different programs were pre-installed on the template so all the deployed virtual machines would be ready to use as soon as they were powered on. The installed programs were an RCS 9500 dispatcher, Python with test automation libraries, OpenSSH, VMware Tools, features on demand (FOD), and needed drivers. OpenSSH was installed for SSH remote connec-

tion. VMware Tools were installed for the ability to get the IP address of the virtual machine using the vSphere Automation API. The firewall was configured to allow inbound TCP traffic from port 22 which is the default port of the SSH [31].

5 Software Development

The development process started after the major guidelines in the planning phase were done. The first step was to create a proof of concept where the requests of the vSphere Automation API were executed with Python code. The API requests were done using the urllib3 library of Python which is an HTTP client providing functionality to make HTTP requests [32].

The proof of concept turned out to be successful and the development of the class responsible for vSphere Automation API requests began. The classes that were developed are VMware, SSH, Airbus software configurator, csv, and virtual machine classes. The initial program was designed to be driven in command line environment and there was no need to develop a separate class for GUI elements.

5.1 VMware Class

The first class that was developed was a VMware class responsible for vSphere Automation API requests. The class is the most important class of the program and it can be used individually or with cooperation with other classes of the program. The VMware class handles user authentication, virtual machine deployment from OVF templates, and power on and off functions for the deployed virtual machines.

The constructor of the VMware class takes the login credentials of the user, the username, and password as parameters. The login function which requests the access token from vSphere is called inside the constructor. If the entered credentials are valid, the function will return an access token which will be stored as a local variable, meaning it is accessible by all the functions in the class. The access token is always needed when new requests will be done using the vSphere Automation API. If the entered user credentials are not recognized by the vSphere, the program will end. The constructor and login function of the VMware class are shown in listing 1.

```
class VMwareClient:

    def __init__(self, username, password):
        self.token = self.login(username=username, password=password)
        self.results = []
```

```
    # Login to VMware. Returns access token.
    def login(self, username, password):
        url ='https://vcenter/rest/com/vmware/cis/session'

        headers  = urllib3.util.make_headers(basic_auth=username+':'+password)
        response = http.request('POST', url, headers=headers)
        response = json.loads(response.data)

        try:
            self.printMessage("Logged in. Access token: "+response["value"])
            return response["value"]
        except:
            self.printMessage("Login failed: " +
            response["value"]["messages"][0]["default_message"])
            quit()
```

Listing 1. The constructor and login function of the VMware class

The virtual machine deployment function deploys virtual machines from OVF, open virtualization format, templates using a POST request from the *vcenter* part of the vSphere Automation API. The request takes the ID of the wanted OVF template as a parameter. The resource pool where the virtual machine will be deployed has to be defined in the JSON format and attached to the body of the request. The Python function takes the ID of the OVF template, the ID of the resource pool, and the name for the virtual machine as parameters. Before the deployment request calls in virtual machine deploy function, other virtual machines in the vSphere are filtered with the name parameter of the virtual machine to avoid duplicates. If the user has the rights to perform deploy actions, access to the content library and the resource pool, and if there are no duplicate virtual machines in the vSphere, the request will be successful. The OVF deploy function is presented in listing 2.

```
# Deploy VM from OVF template to target resource pool
url = "https://vcenter/rest/com/vmware/vcenter/ovf/library-item/id:" + tem-
plateId + "?~action=deploy"
payload = json.dumps({"deployment_spec": {"accept_all_EULA": True,"name":
vmName},"target": {"resource_pool_id": resourcePool}})
response = http.request('POST', url, headers={
    'Content-Type': 'application/json',
    'vmware-api-session-id': self.token
}, body=payload)
http_status = response.status
response = json.loads(response.data)
if (http_status == 200):
    print "Deployment for " + vmName + " completed."
    self.results.append(response)
    return response
else:
    return response
```

Listing 2. The virtual machine deployment function

The library item and resource pool IDs are long random character strings which can be extremely hard to remember. Functions to return these IDs were developed. The return functions initiate simple HTTP GET requests to return the IDs from their corresponding deployed virtual machines. When the virtual machines are deployed to the resource pool, their IDs and IP addresses can be returned. The ID of the virtual machine can be returned by filtering the virtual machines in the vSphere with the name of the virtual machine. When the target virtual machine is found, its ID can be returned with the get_vmId(vmName) function. The IP address request demands that the virtual machine has VMware Tools installed and the virtual machine must be powered on. The deployed virtual machines can be restarted and powered on or off with Python functions when needed.

5.2 Configuration File, Virtual Machine Class and CSV Class

The user needs to create a CSV (Comma-separated values) configuration file to be able to use the software. The configuration file requires specific fields to be included and filled out with correct data. The required data is vSphere and Airbus Defence and Space's software-related. The required vSphere data is the ID of the content library, the ID of the OVF template, the ID of the resource pool, and the name virtual machine.

The virtual machine class saves the configuration information from the CSV configuration file. The constructor of the class takes all the information from the CSV as parameters. The purpose of the class is to make data handling and processing more organized.

The CSV class is responsible for reading the CSV configuration file and saving the information in the file in a virtual machine object. The operation of the CSV class is very simple. The constructor takes the path of the configuration file as a parameter and the CSV file is opened and read in the `getVMlist(self)` function. The function creates an empty virtual machine array and iterates it through all rows of the CSV file in a for-each loop. During every iteration a new virtual machine object is created and the information of the row is stored in the object. After every iteration, the created virtual machine

object is added to the array. When the function has finished, it returns the filled array of virtual machine objects.

5.3 SSH Class

An SSH class was developed for connecting the automation software with the deployed virtual machines. Even though the virtual machines were deployed from templates that contained the required software and configurations, they still needed some after deployment configurations and service management. The software of the Airbus Defence and Space required the IP address of the virtual machine and that could not be obtained before the virtual machine was powered on since the IP address came from the DHCP server.

The constructor of the SSH class takes the IP address of the virtual machine, username, and password as parameters. Python's Paramiko library was used for the SSH connection. An SSH client variable was created inside the constructor and the `connect()` function was called to return the SSH client object. The `connect()` function creates and returns the SSH client with the SSH connection if the target host virtual machine is reachable and user credentials are valid.

The SSH class contains functions for executing commands, uploading, and downloading files. It also has ready-made functions to initialize Airbus Defence and Space's software and services.

5.4 Airbus Software Configurator Module

Airbus software configurator module contains configuration file generation classes for Airbus Defence and Space's software. The classes generate documents in the XML and CSV formats and use information in the configuration CSV file of the software to fill in the blanks.

A wider operation of the classes cannot be disclosed due to confidentiality reasons. The use of the class is optional, and it can be run independently or as a part of the program depending on the use case.

5.5 The Main Method

The main method of the software utilizes all the classes mentioned under the previous subheadings. The complete workflow of the software is described in a flow chart in figure 4.

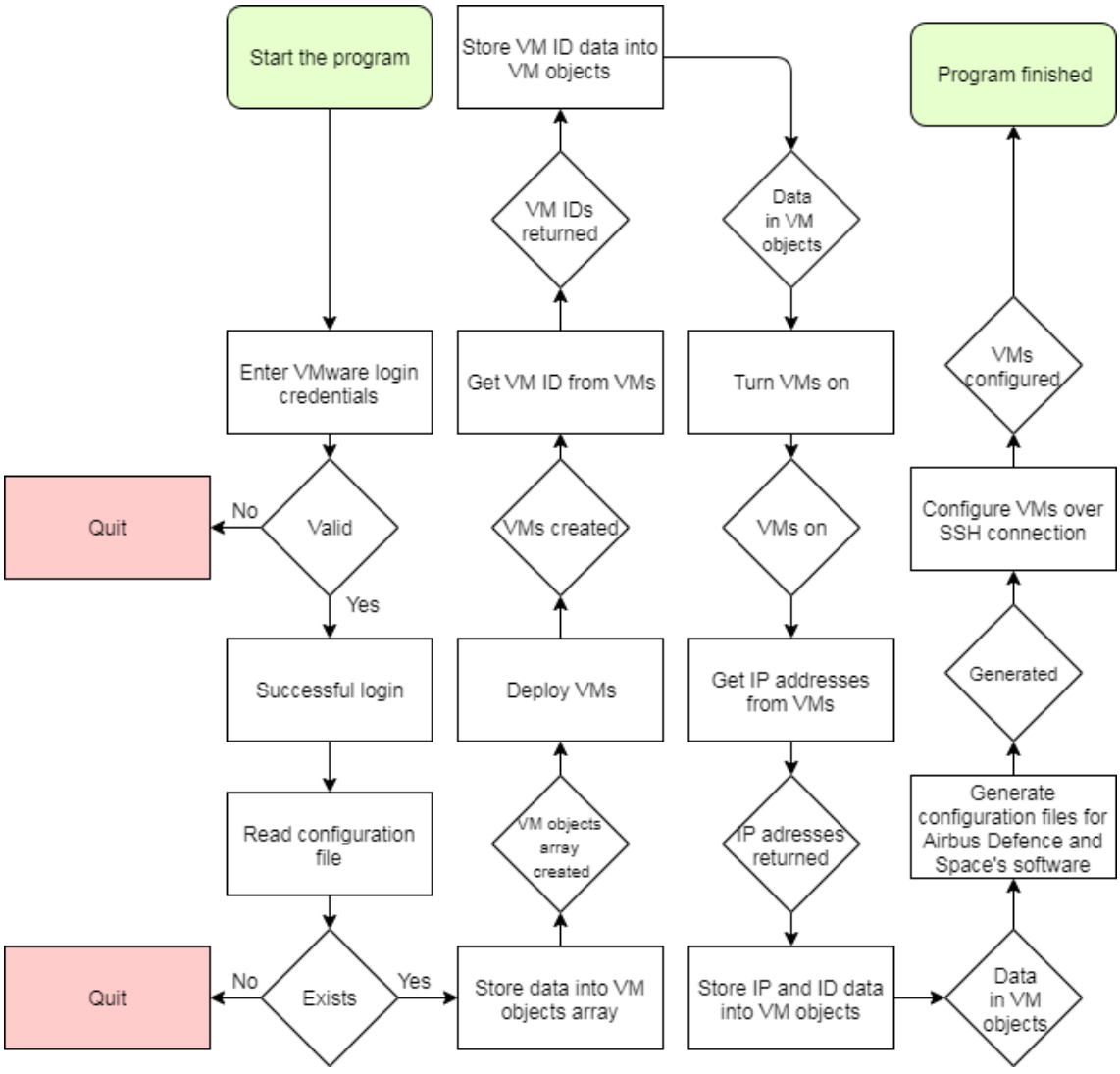

Figure 4. Flow chart displaying the software's workflow

The flow of the program begins by entering the user credentials. The user credential information is used as parameter for the VMware object. The Getpass library of Python is used for the password input to hide the entered characters in the command terminal.

If the user credentials are valid and the user has rights to use vSphere through the vSphere Automation API, the program reads the CSV configuration file and saves the information of the file into the class variables of a virtual machine object. If the file is not available or it is missing information, the program will end.

After the array of virtual machine objects is created, the program will start deploying them using the deploy function of the VMware class. The `deployVirtualMachines(vmList)` function of the main method creates an array of threads using the `deploy_ovf()` function of the VMware class. The threads are started and joined together in a for-loop resulting in a situation in which the virtual machine deployment will start simultaneously in vSphere. Because the threads have been joined, the program will wait for all threads to finish until the program can proceed. The beginning preparations of the program and the virtual machine deployment function are shown in listing 3.

```
username = raw_input("Enter username: ")
password = getpass.getpass("Enter password: ")
vmClient = vmware.VMwareClient(username,password)
filename = "Configuration.csv"
csvClient = csvclient.Client("C:\\ConfigurationCSV\\" + filename)
vmList = csvClient.getVMlist()

# Deploy virtual machines using threads
def deployVirtualmachines(vmList):
    threads = []

    for vm in vmList:
          thread = threading.Thread(target=vmClient.deploy_ovf,
          args=[vm.vmData["template"], vm.vmData["name"], vm.vmData["re-
          sourcePool"]])
          threads.append(thread)

    for thread_i in range(len(threads)):
          threads[thread_i].start()

    for thread_k in range(len(threads)):
          threads[thread_k].join()
```

Listing 3. The deployment function in the main method

When the virtual machines are successfully deployed, the program will request for the IDs and IP addresses of the virtual machines. They are then stored in corresponding virtual machine objects for later use.

The virtual machines are powered on with the `powerVMs(vmList)` function. The function uses the `startVM(vmId)` function of the VMware class to power on the target virtual machine which is found using the ID of the virtual machine. The virtual machines are powered on in batches of ten in a for-loop. There is a short 3-second wait after every power on iteration to prevent CPU overload. After the last iteration, there is a two-minute wait to make sure that all virtual machines are booted. Once all batches are processed and virtual machines are powered on, the program will get the IP addresses which are saved into virtual machine objects as class variables and the program will move on.

In the next phase the Airbus Defence and Space's software configurations are generated and SSH connections are made between the client and virtual machines. The SSH connections are made in batches of ten, in the same way as the virtual machines were powered on. The reason for this is to prevent the network from jamming due to the increased traffic from the SSH connections. After the SSH commands have been executed on all virtual machines, the virtual test environment is setup and the program will finish.

6 Software and Virtual Environment Testing

All vSphere Automation API calls used in the final year project were tested manually using the vSphere Automation API explorer tool and the Postman application to find out what kind of responses the requests gave and what kind of payloads they required for the headers.

The software code was tested during the development cycle by running parts of the code to verify that everything works. Test driven development was not done during the project for test cases were planned and executed after the functions were written.

The performance testing of the code and vSphere played a very important part of the project. The goal of the performance tests was not to test the performance of the software but the performance of the network and the VMware cluster as they were under an extreme load when the software was generating new virtual test environments. The goal was to find the most optimal parameters for the program to achieve maximum efficiency resource-wise. The performance tests were run frequently throughout the project.

Two resource demanding operations were found in the workflow of the software: deploying new virtual machines from a template and powering on created virtual machines. The deploying function was heavy on the network and the cluster while the powering on function mainly stressed only the cluster.

6.1 Virtual Machine Deployment

For the deploying function the goal was to determine if running the functions in multiple threads would be more efficient compared to running them in a normal for-loop. In the multithread approach, the virtual machine deployment tasks were started simultaneously with vSphere, thus creating instantly a great load. The for-loop approach created the deployment task, waited until the virtual machine was created, and then started a new task and repeated this until the environment was done.

Another test was executed to find out if using multiple identical templates located in the same content library would be more efficient than using just one template. This idea was

first introduced by the creator of the core cluster. The theory was that using multiple templates would stress the cluster and the network less if the ESXi servers would not use the same file at the same time [26]. The test setups and results are displayed in table 2.

Table 2. Virtual machine deployment times

	Seconds (s)		
VM quantity	1 Template (thread)	2 Templates (thread)	2 Templates (w/o thread)
1	220	169	244
5	941	805	1,220
10	1,737	1,368	2,440
25	4,322	4,265	6,100
50	9,000	9,415	12,240

The test was to deploy 1, 5, 10, 25, and 50 virtual machines using the thread approach with one and two templates and using the for-loop approach with two templates. The results were surprising as there was almost zero to no difference between the 1-template and 2-template multithread approach. The multithread approach was around 25 % more efficient compared to the for-loop approach. Test results explained why the multithread approach was better. The results are displayed as a line chart in figure 5.

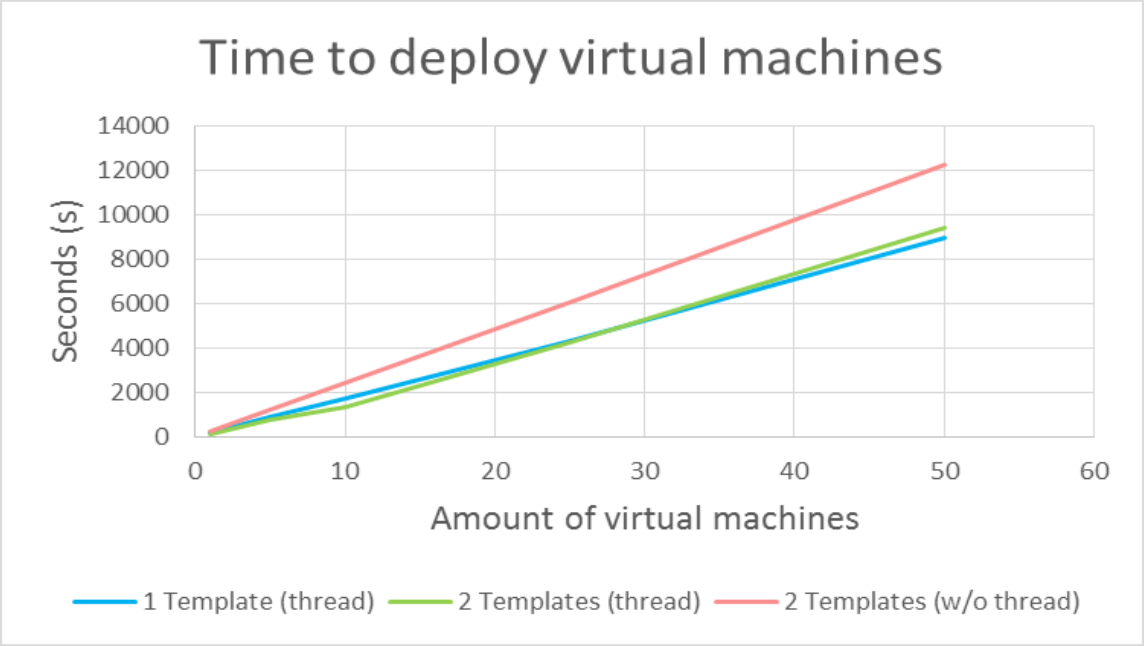

Figure 5. Line chart displaying deployment times

The multithread approach was proven to be 25% more efficient when compared to the normal for-loop. It was found out the vSphere had a built-in event queue that could process ten different tasks at the same time when all processes were given to vSphere. This resulted in a situation where all four ESXi servers of the cluster worked on four different virtual machines at the same time while the for-loop approach only used one server at a time. The tasks were shared between the ESXi servers, and once a task was completed, the next task in the queue was taken into processing.

6.2 Powering on Deployed Virtual Machines

The other important test was to test how powering on all the deployed virtual machines would affect the cluster. It was clear that if all the virtual machines were booted at the same time using threads, it would overload the resource pool or the whole cluster. The overload was expected even if the VMware had an event queue of ten simultaneous actions since the power on task is reasonably fast to complete. The power on process will always peak the CPU load and in our case the goal was to boot 255 individual virtual machines. The risk of overload was certain, so a controller power on method was developed.

The method was to power up all virtual machines in small batches. The optimal batch size which was used was ten. The process was to start ten individual virtual machines, wait until the vSphere Automation API gave a response and wait for an extra 20 seconds until the next ten virtual machines are taken into the process. The best approach would have been to make the power on process dynamic by requesting the CPU and RAM loads of the resource pool with the vSphere Automation API and to start new virtual machines as the peak load has decreased. Unfortunately, the API lacked this kind functionality.

The theoretical time to power on 255 virtual machines was about 1 hour and 10 minutes. That is the time the power up process would take in a perfect situation where there are no other services running in the cluster. This situation is displayed in figure 6.

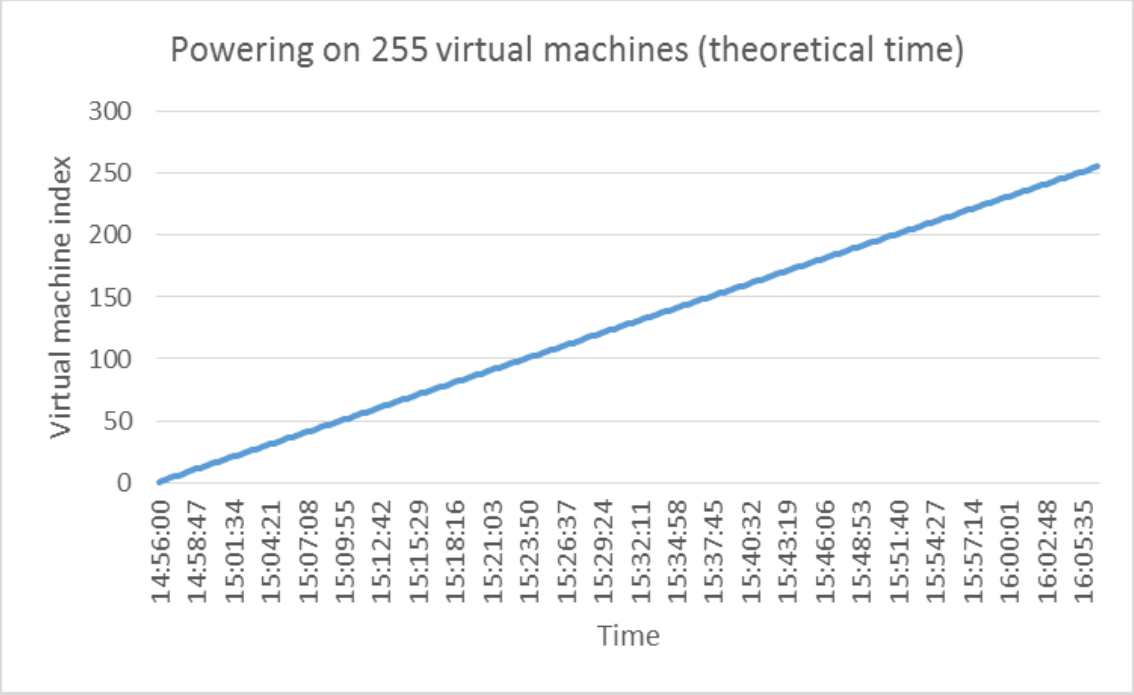

Figure 6. Theoretical time to power on 255 virtual machines

The realistic result for the task to complete was roughly 3 hours. The CPU load was under control and no other services were interrupted during the power on sequence. The CPU load peaked gradually but became steady after all virtual machines were on. The results with the CPU load and total time can be found in figure 7.

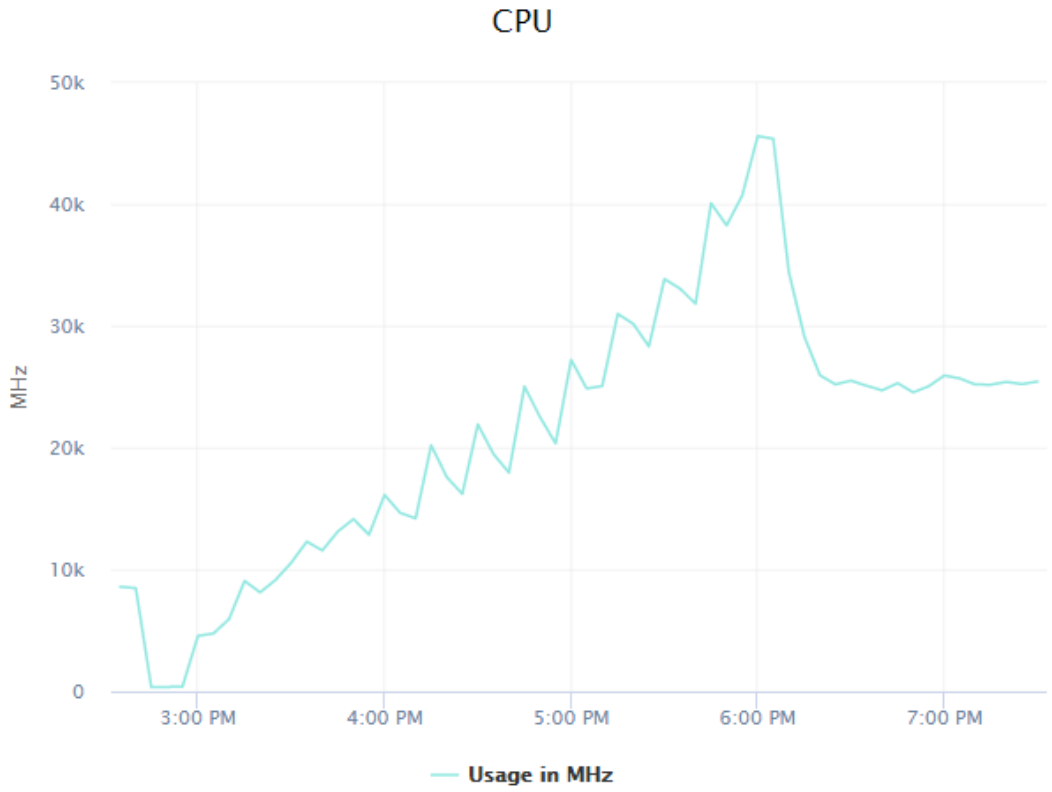

Figure 7. Actual time to power on 255 virtual machines with CPU load

The test showed that the power on process will not always be calculable beforehand. Other processes were run in the cluster at the same time and they may have caused the delays.

6.3 SSH Connections

SSH connections were tested to find out if it was possible to have a simultaneous connection from the client to all 255 virtual machines at the same time. The test was to create 255 individual SSH objects and run initialization commands over SSH using threads. The SSH threads were started at the same time and the connections started to appear. Soon it was noticed that the connections were not as reliable as required. Some commands went through with ease and some did not result in misconfigured virtual machines.

It was decided to give up on the idea of running the SSH connections with threads and to take the more controllable approach by using a normal for-loop. The for-loop approach

was tremendously slower than the thread approach but with it everything could be monitored that was happening in the program much better. The for-loop approach eventually worked without dropping any SSH connections, so it was decided to use it in the final product.

7 Software Implementation

The software was created to be a tool for integration and verification teams. The purpose of the software was to create ready-to-use virtual test environments on the vSphere platform.

At first, the virtual test environment generation was used to create a test environment of 255 individual virtual machines. The virtual machines were used to conduct capacity tests for Airbus Defence and Space's RCS 9500 dispatcher solution. The tests were run successfully but the need for the future use of the software remained.

Due to the flexible nature of the software, it can be used to create virtual environments of any size. The only limiting factors are vSphere and network infrastructures. The deployment times need to be taken into consideration when large quantities of virtual machines are deployed. The deployment times are linear and when quantities grow, the deployment time grows.

Even though the software was developed for integration and verification teams who use it for testing, it can also be used to create virtual environments for other purposes. At the time of writing this thesis, the software has been used as a tool to create a configured virtual environment for another project and its future potential has been acknowledged.

8 Conclusion

The purpose of this thesis was to develop a solution for Airbus Defence and Space Oy which can automate virtual test environment creation, configuration, deletion, and management on VMware vSphere platform. The solution was to be compatible with other technologies used by the company and it needed to be scalable for future use.

The developed software was proven to be a useful tool for generating configured virtual test environments for integration and verification teams. The value that the software brings can be measured both in time and money. If the virtual environments had been created on traditional environments using dedicated hardware or if the virtual machine deployment and configuration had been done manually, the costs would have been huge. The solution helped the company to save resources such as time, energy, and workhours.

Even though the software turned out to be useful, well planned, and executed, there is still room for future improvements. The performance of the test environment creation automation process could be improved drastically in the future with some specific changes.

The VMware vSphere Automation API does not support virtual machine cloning at the moment, but it is possible that it will be added to the API in the client company in the future. When this happens, it is recommended to implement the cloning feature into the code to make the deployment process faster if the VMware environment is still being used.

The best choice, performance wise, would be to create the whole test environment creation process in the Docker environment. The Docker environment is more efficient and saves more time and computational resources compared to the VMware solution [25]. This option will be available if driver related issues are solved either in Airbus Defence and Space's software or in Docker.

References

1 Rouse, Margaret. 2019. Online. Virtualization. https://searchservervirtualization.techtarget.com/definition/virtualization. Accessed January 8, 2019.

2 Burger, Thomas. 2012. Online. The Advantages of Using Virtualization Technology in the Enterprise. https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise. Accessed January 14, 2019.

3 Airbus. Online. Airbus in Finland. https://www.airbus.com/content/dam/channel-specific/website-/company/global-presence/Airbus-in-Finland.pdf. Accessed January 14, 2019.

4 Hiltunen, Jukka. 2010. Palvelimen virtualisointi. Bachelor's Thesis in telecommunications technology. https://www.theseus.fi/bitstream/handle/10024/24102/Hiltunen_Jukka.pdf?sequence=1&isAllowed=y. Accessed January 8, 2019.

5 Rouse, Margaret. 2016. Online. Virtual Machine (VM). https://searchservervirtualization.techtarget.com/definition/virtual-machine. Accessed January 8, 2019.

6 Portnoy, Matthew. 2016. Virtualization Essentials, Second Edition. John Wiley & Sons, Inc. Accessed January 16, 2019.

7 Kivisaari, Tero. Online. API:t ovat modernin integraatiostrategian ydin. https://blog.digia.com/rest-api. Accessed December 11, 2019.

8 REST API Tutorial. 2018. Online. What is REST. https://restfulapi.net/. Accessed January 12, 2019.

9 Ylönen, Tatu. 2019. Online. SSH.COM. https://www.ssh.com/ssh/?utm_source=s&utm_medium=nav&utm_campaign=head. Accessed December 8, 2019.

10 Maheshawari, Mudit. 2017. Online. Understanding SSH Workflow. https://medium.com/@Magical_Mudit/understanding-ssh-workflow-66a0e8d4bf65. Accessed December 11, 2019.

11 Bisson, David. 2019. Online. How Secure Shell (SSH) Keys Work. https://www.venafi.com/blog/how-secure-shell-ssh-keys-work. Accessed December 11, 2019.

12 VMware. 2019. Online. VMware Timeline. https://www.vmware.com/timeline.html. Accessed January 6, 2019.

13 VMware. 2019. Online. VMware Products. https://www.vmware.com/products.html. Accessed January 6, 2019.

14 Rouse, Margaret. 2017. Online. VMware vSphere. https://searchvmware.techtarget.com/definition/VMware-vSphere. Accessed January 12, 2019.

15 VMware. 2019. Online. vSphere. https://www.vmware.com/fi/products/vsphere.html. Accessed January 12, 2019.

16 VMware. 2009. Online. Introduction to VMware vSphere. https://www.vmware.com/pdf/vsphere4/r40/vsp_40_intro_vs.pdf. Accessed January 6, 2019.

17 Rouse, Margaret. 2019. Online. VMware ESXi. https://searchvmware.techtarget.com/definition/VMware-ESXi. Accessed January 12, 2019.

18 Lakmal, Aruna. 2019. Online. What is VMware vCenter Server? http://www.mustbegeek.com/what-is-vmware-vcenter-server/. Accessed January 12, 2019.

19 VMware. 2019. Online. vSphere Automation API 6.5. https://code.vmware.com/apis/191/vsphere-automation. Accessed January 6, 2019.

20 Python. 2019. Online. PEP 373 – Python 2.7 Release Schedule. https://www.python.org/dev/peps/pep-0373/#update. Accessed January 6, 2019.

21 Microsoft. 2018. Online. What is the Server Core Installation Option in Windows Server? https://docs.microsoft.com/en-us/windows-server/administration/server-core/what-is-server-core. Accessed January 6, 2019.

22 OpenSSH. 2019. Online. OpenSSH Homepage. https://www.openssh.com/. Accessed January 6, 2019.

23 Microsoft. 2019. Online. OpenSSH in Windows. https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview. Accessed January 6, 2019.

24 OpenSSH. 2019. Online. OpenSSH Users. https://www.openssh.com/. Accessed January 6, 2019.

25 Brown, Frank. 2019. Online. VMware vs Docker Comparison. https://rancher.com/blog/2019/a-comparison-of-vmware-and-docker/. Updated December 5, 2019. Accessed December 14, 2019.

26 Koski, Juuso. 2019. Test Systems Engineer, Airbus Defence and Space Oy, Helsinki. Interview. December 16, 2019.

27 VMware. 2019. Online. Distributed Resource Scheduler, Distributed Power Management. https://www.vmware.com/products/vsphere/drs-dpm.html. Accessed December 16, 2019.

28 Triana, Michel. 2012. Online. Library Oriented Architecture. http://www.micheltriana.com/blog/2012/04/09/library-oriented-architecture. Accessed December 17, 2019.

29 VMware. 2017. Online. Cloning Virtual Vachines in vCenter Server (1027865). https://kb.vmware.com/s/article/1027865. Accessed December 17, 2019.

30 VMware. 2019. Online. Working with Templates and Clones in the vSphere Client. https://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.vsphere.vm_admin.doc%2FGUID-F40130B0-0194-4A41-91FA-1A967721924B.html. Accessed December 17, 2019.

31 Ampalam, Manoj. 2018. Online. Install Win32 OpenSSH. https://github.com/PowerShell/Win32-OpenSSH/wiki/Install-Win32-OpenSSH. Accessed December 17, 2019.

32 Petrov, Andrey. 2019. Online. urrlib3 User Guide. https://urllib3.readthedocs.io/en/latest/user-guide.html. Accessed January 7, 2019.