

Satakunnan ammattikorkeakoulu Satakunta University of Applied Sciences

JUHA-MATTI KAUNISMÄKI

Automation of Application Server Installation

BACHELOR'S DEGREE PROGRAMME IN INFORMATION AND COMMUNICATION TECHNOLOGY 2022

Author Kaunismäki, Juha-Matti	Type of Publication Bachelor's thesis	Date August 2022			
	Number of pages 29	Language of publication English			
Title of publication Automation of Application Server Installation					
Degree Programme Bachelor's Degree in Information and Communication Technology					
Abstract					
In this thesis we tried to find ways to automate production of application servers in the VMware server environment. Automating our work made sure that human errors did not appear and that every project came out similar. This helped with debugging problems and other people understanding our part in the project. Often in a real work environment your co-workers might not know what you have done when it comes to configuring application servers manually.					
First, we went over the necessary knowledge to understand what virtualization is and why it is necessary. Then we investigated what VMware could do in this regard to make virtualization easier. We looked deeper into vSphere's two main components, ESXi and vCenter server appliance.					
After that, we investigated application servers and their purpose in this environment.					
Learned how important infrastructure as code is in our lives and how both Terraform and Ansible made it easier to accomplish our goal.					
We prepared our Linux running on Windows 10 with the necessary installations. The next steps required were to get Terraform to create the virtual machine and Ansible to run our playbook on it. We further investigated the structure of these two programs and did our ansible playbook run.					

Keywords Virtualization, Automation, Virtual Machine, Linux, Hypervisor, Server, Application Server

FOREWORD

I would like to thank Cimcorp Oy at Ulvila for the opportunity to do my thesis for them and all my co-workers for their help in making this possible. I would also like to give my sincerest thanks to my teacher Juha Aromaa for his patience and help during my studies. Lastly, I would like to thank everybody who helped in the creation of this thesis.

CONTENTS

1 INTRODUCTION	6
2 VIRTUALIZATION	8
2.1 Virtualization in Servers	8
2.2 VMware vSphere	9
2.2.1 ESXi Hypervisor	10
2.2.2 VCenter Server Appliance	11
2.3 Application Server	
3 INFRASTRUCTURE AS CODE	13
3.1 Ansible	13
3.1.1 Modules and Syntax	15
3.1.2 Playbooks and Inventory	15
3.2 Terraform	17
3.3 Why Use Both Ansible and Terraform Together?	19
4 PRACTICAL WORK	
4.1 Preparations	
4.2 Creating Virtual Machines with Terraform	
4.2.1 Operating System and Templates	
4.2.2 Terraform Run and Ansible Integration	24
4.3 Ansible Roles Configuration	25
4.3.1 Playbook Structure	
4.3.2 Playbook Run	
4.4 Results	
5 CONCLUSIONS	
REFERENCES	

LIST OF SHORT TERMS

HCL: HashiCorp Configuration Language is HashiCorp's own language that runs natively on Terraform.

HTML: HyperText Markup Language is the standard web language for most websites.

HTTP: HyperText Transfer Protocol is an information transfer protocol on world wide web.

Idempotent: Operation where you run the same inputs on target, and it does nothing if the state of the target has not changed. A big part of Ansible.

JSON: JavaScript Object Notation is easy to read and write programming language.

SSH: Secure Shell is a network protocol to connect two computers over secure shell connection. For example, to use remote login and run command on remotely.

VM: Virtual Machine is a virtualized operating system that runs on top of physical hardware like server or laptop.

WinRM: Windows Remote management is Microsoft's own remote management tool to run PowerShell on target and do much of the same as SSH.

WSL: Windows Subsystem for Linux Native compatibility layer on Windows that allows you to run Linux natively on Windows 10, 11 and Server 2019.

YAML: Yet Another Markup Language is markup language for data. Human-readable and easy to understand programming language.

1 INTRODUCTION

This thesis was made in attempt to understand and improve IT infrastructure automation at Cimcorp Oy. We will investigate how this is achieved using tools such as Ansible and Terraform in tandem. This will happen by creating virtual machines that run specific application servers on them.

Automating your IT infrastructure is becoming increasingly important nowadays. The reason for this is ever more complex and big IT infrastructures. Focusing the limited resources that companies have for maintaining such big systems is an important reason for automating as much as you can. The benefits you get from large scale automation and the speed in the production environment are crucial factors why it is good for companies. Automation makes it possible to expand your environment and reduce costs for companies. It also helps to get rid of human errors while creating the infrastructure for application servers. This is one of the best reasons for any company to think about further automating their infrastructure. (VMware, 2022, What is infrastructure automation?)

Automating IT infrastructure goes hand in hand with virtualizing servers. Virtualization of servers is a major step in getting rid of limitations of physical hardware in an ever increasingly connected world. It helps with the speed of creating such systems and maintenance when you connect to servers from far away. Companies like Cimcorp which have projects going around the world connecting remotely is particularly important.

Working at Cimcorp was wonderful opportunity to gain experience in all kinds of new skills. For instance, a major part of this thesis is running virtual machines on VMware's vCenter servers in a production environment. Using Ansible and Terraform on said virtual servers to understand the creation of the different virtual machines needed to finish a project.

We are going to use VMware vSphere 6.7 for this thesis and Ubuntu 20.04 running on Windows 10 as our configuration Linux. The Virtual Machines will have Red Hat Enterprise Linux version 8.6 on them. At the time of writing, Ansible version is 2.9 and Terraform is 1.2.7. We will get to these in more detail at the practical part of this thesis and the theory.

2 VIRTUALIZATION

Virtualization in general means when physical systems get rid of the hardware limitation and imitate the physical thing on virtual level. This opens many possibilities for all kinds of different applications, not just virtual machines. We will look at later in this thesis. When talking about virtualization on the IT side this means getting rid of specific servers doing just one task for example. Virtualizing these servers, you can run the same tasks on just one or two hardware servers or on the cloud. In practice, this means sharing same hardware resources for multiple virtual machines using hypervisors. We will get into hypervisors later in this thesis. (Red Hat, 2018, What is virtualization?)

Benefits of virtualization are what drove the change from physical hardware to virtualization. Some benefits are for example a more effective allocation of resources. Nowadays, IT does not need to buy different servers for every different operating system they want to run or application. This saves the company money in the long run. Another reason is managing and connecting to said resources. These virtual machines are easy to manage on a virtual environment and can be connected from anywhere. If said VM for example needs to be rebooted or crashes, it can be brought back online faster as a virtual version than physical which means less downtime in services. Lastly, you can scale your infrastructure faster and provision it with tools like Ansible and Terraform. (IBM, 2019, Virtualization-a-complete-guide.)

2.1 Virtualization in Servers

Server virtualizations tried to solve the problem when running multiple different operating systems or applications on one hardware. Each server requires their own hardware to work. When virtualizing said server, you can use both Windows and Linux servers on the same hardware. They all work like they have their own hardware despite sharing the same resources. This is achieved using a software layer called hypervisor. There are many different hypervisors from different companies on offer like ESXi that we use in this thesis. (Strickland, J, 2008, How server virtualization works) There are two different types of hypervisors. Type 1 which is often called bare-metal hypervisor. These are such products as Kernel-Based Virtual Machine (KVM) on Linux, Microsoft Hyper-V, Citrix and ESXi. Type 2 hypervisor on the other hand needs its own host to run on to. This is usually an existing operating system. In this work, we will be using Type 1 hypervisor ESXi.

(Marshall, N., Brown, M., Fritz, G. B., & Johnson, R, 2018, Mastering VMware vSphere 6.7)

2.2 VMware vSphere

VMware's vSphere is the main concept that drives this thesis. It is a collection of VMware's multiple tools that you use to virtualize your infrastructure. Two main parts of what makes it work are ESXi and vCenter Server. With these two, we can create a virtual system that can run the needed virtual machines and applications. (VMware, 2022, VMware vSphere documentation.) There are other products and functions that make the whole package but these two are the most important ones to understand.



Picture 1. How vSphere works. (VMware, 2022)

2.2.1 ESXi Hypervisor

ESXi is a Type 1 hypervisor that runs on the physical server. This is run by VMkernel which is Linux based. It is installed directly on the server and handles the server's host. As we can see in picture one, ESXi handles the layer between the virtual machines and physical hardware. We can connect to the ESXi through the direct service console (Picture 2) and assign to it an IP address. This way we can connect to it with a browser from anywhere as long we are in the same network. This is achieved using the host client site. (Picture 3) (Seaton, J, 2011, What is VMware ESXi server?) We can also run Linux command line on this direct service console if needed for debugging.



Picture 2. Main page of ESXi 6.7.0 direct service console.

nware ESXI						
Navigator 🛛	esxi					
itest Manage Monitor * Ø Virtual Machines • ⑤ Vortual Machines • ⑤ Storage • ⑥ Networking	Manage with vCenter Server State Uptime:	Create/Repatre VM 200 Shut down 200 Reboot 4.7.5 Updata 3 Shut 1995723) Hammal (connected to sCenter Server at 1 47.12 days	C Refresh 🏠 Actions			
	✓ Hardware					- Configuration
	Manufacturer	HPE				Image profile
	Model	ProLiant DL380 Gen10				vSphere HA state
	P E CPU	8 CPUs × Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz				► vMotion
	Memory	111.68 GB				✓ System Information
	Resistent Memory	0 B				Date/time on host
	Virtual flash	0 B used, 0 B capacity				Install date
	 Metworking 					Asset tag
	Hostname	esxi				Serial number
	IP addresses	1. vmk0: 10. 2. vmk1: 10.				BIOS version
	DNS servers	1. 10.				DIGG TEREASE GATE
	Default gateway	10.				 Performance summary last hour
	IPv6 enabled	No				
	Host adapters	6				100
	Networks	Name			VMs	3 80
		Q DRBD			4	00
		Q VM Network			9	0 60
		VM Network 2			0	ž 40
		VM Network 3			5	5
	✓ I Storage					8 20
	Physical adapters	3				mm
	Datastores	Name	Туре	Capacity	Free	12:07 12:11
		-hddraid6	VMFS6	1.31 TB	496.53 GB	
	Recent tasks					
	Task	~ Target		~ Initiator	~ Queued	✓ Started

Picture 3. Summary page of ESXi 6.7 U3 host client.

2.2.2 VCenter Server Appliance

VCenter Server Appliance (VCSA) is a premade Linux virtual machine that runs the VMware vCenter server on it. This virtual machine works as the centralized server for all the other virtual machines that the user wants. It is installed on the ESXi host directly and works on top of it. The main reason to use vCenter is if there is more than one ESXi host. VCenter works as the managing and monitoring site for all the necessary virtual applications running on it. It can also perform other functions like make templates out of VMs and manage the ESXi hosts that is running on. Other key features are managing the virtual switches and network interface controllers (NIC) of the server.

(Marshall, N., Brown, M., Fritz, G. B., & Johnson, R, 2018, Mastering VMware vSphere 6.7)

2.3 Application Server

Putting the application server into context is hard because of how varied and wide the subject is. Generally speaking, we are talking about servers that provide the server framework for different applications to run on. Application servers are usually located between the physical server hardware and the database. Their main tasks are to provide the business logic for the applications and to query information from the database or database server. Application servers are often installed on virtual machines or containers and create dynamic web content on site, pulling that info from the database. Unlike web servers that only creates static HyperText Markup Language (HTML) web content. Sometimes these two work together but often times the line between web server and application server is thin. (Ingalls, S, 2021, What is an application server?)

Basic example of (Picture 4) how the application server could work is that first the user accesses a certain web page on their browser. They then send out HyperText Transfer Protocol (HTTP) inquiry to the web server and the web server asks the application server for help in creating dynamic pages. This is when the application server sends out inquiry on the database for example. (Phipps, J, 2022, Web servers vs application servers: What's the difference?)



Picture 4. Basic idea of Application server in use. (Webopedia, 2021)

3 INFRASTRUCTURE AS CODE

To automate a lot of infrastructure you need to understand certain basic concepts like infrastructure as code. (IaC) It is a new idea in the IT automation world which only came to be coined as a term in the early 2000s. Before that, the idea existed in the 1990s to automate IT with scripts which are still used as part of IaC. (Carey, S. 2021, What is infrastructure as code?)

Then what is the idea of infrastructure as code? It is quite simple at its core. Everything is code and getting rid of the manual configurations to set up IT servers and VMs is the key to saving time for more important tasks. To get reliable and repetitive infrastructure you need to create it from small easy to handle pieces like modules in Ansible to make code for the infrastructure. This is usually achieved using languages like YAML or JSON because they are easy to understand and can easily be changed for the needs of the new infrastructure. Making these changes at the code is safer than doing them at the target especially if it is already in production. (Carey, S. 2021, What is infrastructure as code?)

The benefits that come from using IaC to create your environments is easy to see. It is faster, safer, and more cost-effective for IT teams to deploy and test their VMs. Using a provisioning tool like Terraform and a configuration tool like Ansible makes it easy and safe to test your new ideas at the local servers or even on your own laptop. (Carey, S. 2021, What is infrastructure as code?)

3.1 Ansible

Ansible is an open-source automation IT tool sponsored and maintained by Red Hat. The community also contributes a lot. It is agentless which means it does not need any service or daemon to run on the user's operating system of choice. Being agentless makes it easy to use on most systems, servers or PCs. The only requirement that it has on the operating system side is that the Python interpreter needs to be installed and that comes default in most Linux distributions. (Verona, J, 2016, p. 120) Running Ansible on Windows is a bit more complicated since Windows does not have native Ansible support but we will get into Windows Subsystem for Linux (WSL) later in the practical part of the thesis.

One benefit of using Ansible to configure your IT infrastructure comes from the communication between the user and the target. Ansible uses Open Secure Shell (OpenSSH) in most cases when dealing with Linux and Windows Remote Management (WinRM) when dealing with Windows servers. (Oh, D., Freeman, J., & Locati, F. A. 2020, p. 10) This makes it easy and safe to connect to multiple targets at the same time and running needed tasks on them. We will get more into how Ansible connects over SSH in the practical part of the thesis also how to install and operate Ansible.

Ansible is declarative which means its user declares the state it wants the target to be and Ansible tries its best to configure it that way. Compared to the traditional way where you need to understand coding and the logic behind it, Ansible takes away this problem. This means you can run Ansible on the target as many times without breaking anything because if the target is already in the desired state, then Ansible will not make any changes. Operation like this is officially called idempotent. Being declarative also gives the benefit that most commands only work on certain operating system like Windows. If you instead use generic commands, then you can configure different OSs at the same time. (Smith, S. R. & Membrey, P. 2022, Chapter 2)



Picture 5. Key concepts that make Ansible such a powerful tool. (LinkedIn, 2020)

3.1.1 Modules and Syntax

To understand how Ansible works you need to understand how it delivers the tasks and commands to the target. In this, Ansible uses a clever system called modules which themselves are like small "programs." (Ansible & Red Hat, 2022) Modules can also be called "task plugins" or "library plugins" as Red Hat themselves puts it. (Ansible, 2022, Introduction to modules)

In an actual sense, modules are just small parts of simple YAML code. They are easy to write and understand even by new people trying to get into writing code. (Ansible, 2022, YAML Syntax) Ansible has many different modules meant for different tasks. Modules are the reason Ansible is idempotent as we discussed earlier.

As a side note, Ansible can also run ad hoc commands, but they are not relevant to this thesis right now. Instead, we will soon investigate playbooks that make Ansible the great automation tool it is.

In picture six, you can see the basic concept of YAML syntax. It is simple and easy to read. The only part you really need to worry about is that every YAML file starts with --- and make sure the spaces in the front of the text are correct. Last thing is to make sure there are no empty spaces after certain parts. (Ansible, 2022, YAML Basics)



Picture 6, Simple example how YAML syntax works in Ansible modules.

3.1.2 Playbooks and Inventory

Playbooks are what makes Ansible special. They are in simple terms a collection of tasks or just one task running the modules we discussed earlier. It runs these tasks based on the inventory file. This file has all the target hosts the user wants.

Playbook takes the modules as tasks and runs them on the hosts. This is how you can configure even big infrastructures in no time at all. (Smith, S. R. & Membrey, P. 2022, Modules and Tasks)

These plays use the same YAML syntax as the modules we discussed earlier because they are technically the same thing. Playbooks start their play from the top of the code and run until they complete the tasks needed on target or fail for some error in the code. It does this one task at a time and by understanding Ansible you can use this to make sure correct tasks run on correct hosts. (Ansible, 2022, Intro to playbooks)

```
* updatewebserve....
1 ---
2 #Example of Playbook
   name: Update web servers
    hosts: webservers
    remote_user: root
    tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.dnf:
10
        name: httpd
11
        state: latest
    - name: Write the apache config file
12
      ansible.builtin.template
         src: /srv/httpd.j2
14
        dest: /etc/httpd.conf
17 - name: Update db servers
    hosts: databases
    remote_user: root
    tasks:
    - name: Ensure postgresql is at the latest version
      ansible.builtin.dnf:
23
        name: postgresql
state: latest
24
25

    name: Ensure that postgresql is started

26
      ansible.builtin.service:
27
28
         name: postgresql
         state: started
29
```

Picture 7. Example of basic playbook and its syntax.

Inventory is another important part of what makes Ansible work efficiently. It is at its core just a simple file either in INI or YAML format that contains all the needed hosts or managed nodes that we have so far called targets. It can contain multiple groups of hosts or just single one. This is where the playbook gets its hosts: part of the play. We will get into how inventory works in the practical part.

(Ansible, 2022, How to build your inventory)





Picture 9. How Ansible works on multiple hosts. (tutorialspoint, 2022)

3.2 Terraform

Terraform is an open-source infrastructure orchestration tool made by HashiCorp. It can be used to provision cloud and local resources for the IT infrastructure needs. In this work we will focus on the local part but Terraform is especially adept at provisioning cloud-based resources like AWS, Azure, and Google Cloud. Terraform uses its own HashiCorp Configuration Language or HCL for short. It is a unique language made by HashiCorp themselves, but it is visually like JavaScript Object Notation. (JSON) Being a declarative language means the state of what the user wants from the target can be told to Terraform. This makes it easy to understand even by newcomers. (Howard, M. 2022, Terraform -- Automating Infrastructure as a Service.) Terraform uses normal text files ending in .tf or .tf.json for JSON syntax. (HashiCorp, 2022, Files and directories - Configuration language)



Picture 10. Basic principle of Terraform. (HashiCorp, 2022)

Providers and Syntax

Providers are a big reason Terraform works so well with different cloud and other IT infrastructure providers. Providers are official plugins for different companies like VMware, Amazon, and Microsoft. They make it easy to run Terraform on their platform of choice. You can also make your own provider and submit it to HashiCorp as community provider. (HashiCorp, 2022, August 9, Providers)

We explored earlier that Terraform uses its own unique language called HCL that is similar to JSON. Now we will look more closely into the syntax of this language. Three main parts of HCL are expressions, arguments and the block structure that uses them. Arguments are short parts that give a value to certain part of the code. Blocks on the other hand use groups of these arguments to deliver what the user wants. Block starts { and ends in }. This way you can put blocks inside other blocks and create more complex code. (HashiCorp, 2022, Syntax) Expressions are the values inside the argument or values for other expressions and they are the third part of the syntax.

(HashiCorp, 2022, Overview - Configuration language) Expressions can be many things like numbers, strings of text, lists, Boolean value (true, false) etc. They can also be null if needed. (HashiCorp, 2022, Types and values - Configuration language)



Picture 11. Shows argument inside a block.

3.3 Why Use Both Ansible and Terraform Together?

Based on the earlier theory we can say this. Considering how similar Ansible and Terraform seem to be, they work better together than alone. As we saw Ansible is a configuration tool and Terraform is a provisioning tool. In the practical part of this thesis, we will see how this works. Overall, Terraform works best on a big scale creating the required infrastructure. Ansible then configures these specific resources on the virtual machine level. This way both program's strengths work together to get rid of repetition in IT work. This makes manual intensive tasks easy and fast.

4 PRACTICAL WORK

Now we will get to the actual reason why this thesis came to be. We will try to see if we can further automate Cimcorp's virtual machines running application servers. This is to see if we can make them more dependable with less human errors. That way, we reduce the required time and manual effort to make application servers. I will not be showing all parts of the process because most of it is in company production and not open to the public. What I can show is to prove that automation of IT infrastructure is possible and to give an idea on how it can be achieved.

4.1 Preparations

First thing we need to do on our laptop running Windows 10 is to make sure Windows subsystem for Linux has been installed and that it is running your choice of Linux distribution. In this we do the following. Open PowerShell as admin and type these commands in it.

- wsl –install
- dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
- wsl –set-default-version 2
- wsl –install -d Ubuntu-20.04

After all that, reboot your Windows a few times. You should have Ubuntu running on your Windows virtually. We are doing this because Ansible and Terraform run natively on Linux. Linux is the production operating system in most companies like Cimcorp.

After you have WSL installed you need to make sure your Linux is up to date. Then you will install Ansible and Terraform. In this thesis, I am going to use Ubuntu 20.04 as the operating system for the necessary programs. I also recommend installing MobaXterm which I am using in this work for easier WSL use on Windows 10. It is not needed for the command line work but useful. Other useful programs on Linux side are nano and tree for example.

Next, we make sure Ansible and Terraform are installed on Ubuntu. Update and install them using these commands:

- sudo apt update
- sudo apt upgrade
- sudo add-apt-repository --yes --update ppa:ansible/ansible
- sudo apt install ansible
- ansible --version (This checks the ansible is version)
- sudo apt update && sudo apt install terraform
- terraform version

Next, we make an extra user that is to run playbook passwordless over SSH connection from the WSL. Use ssh-keygen command on this extra user to generate public SSH keys for it. These will be used later for the target host.

4.2 Creating Virtual Machines with Terraform

We run Terraform against vCenter 6.7 in a local secure development network. You do this on your local WSL by first making your terraform configuration. This includes what we want from the target machine. It can even be just one main.tf file in a directory but it is better to use some variable files with it to make it more modular. This way, you can use some arguments as variables which helps with making a script to automate running Terraform commands on WSL.

Main commands for Terraform:

- Init Prepare your current directory for other commands
- Validate Check whether the configuration is valid
- Plan Show changes requited by the current configuration
- Apply Create or update the infrastructure
- Destroy Destroy previously created infrastructure

These commands of course require configuration to run against. For this we will use a simple main.tf (Picture 12) file that has the VM configuration and provider.tf (Picture 13) to connect to vCenter. The provider file will use terraform.tfvars (Picture 14) as its

source for passwords and the main file will use vars.tf (Picture 15) for necessary info it gets from the Bash script that runs Terraform.

```
data "vsphere_datacenter"
                                     "dc"
     name = "vcsa"
3 }
5 data "vsphere_datastore" "datastore" {
    name
                     = "hddraid6"
     datacenter_id = data.vsphere_datacenter.dc.id
8 }
10 data "vsphere_compute_cluster" "cluster" {
    name
                      = "Cluster
     datacenter_id = data.vsphere_datacenter.dc.id
13 }
14
15 data "vsphere_network" "network" {
                     = "VM Network 3"
    name
     datacenter_id = data.vsphere_datacenter.dc.id
17
18 }
19 #Change name part to suit your template name.
20 data "vsphere_virtual_machine" "template" {
                      = "RHEL8.6_SA_Template"
21
    name
     datacenter_id = data.vsphere_datacenter.dc.id
22
23 }
24
25 resource "vsphere_virtual_machine" "vm" {
26
                           = var.hostname
    name
27
     resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
    datastore_id = data.vsphere_datastore.datastore.id
firmware = data.vsphere_virtual_machine.template.firmware
num_cpus = data.vsphere_virtual_machine.template.num_cpus
     num_cores_per_socket = data.vsphere_virtual_machine.template.num_cores_per_socket
31
     memory = data.vsphere_virtual_machine.template.memory
guest_id = data.vsphere_virtual_machine.template.guest_id
scsi_type = data.vsphere_virtual_machine.template.scsi_type
     memory
     network interface {
       network_id = data.vsphere_network.network.id
40
     disk {
       label
                              = "disk0"
42
43
       size = data.vsphere_virtual_machine.template.disks.0.size
eagerly_scrub = data.vsphere_virtual_machine.template.disks.0.eagerly_scrub
thin_provisioned = data.vsphere_virtual_machine.template.disks.0.thin_provisioned
44
46
47
48
     clone {
        template_uuid = data.vsphere_virtual_machine.template.id
49
50
        customize {
          linux_options {
52
53
             host_name = var.hostname
                         = "
             domain
             time_zone = "Europe/Helsinki"
55
          }
56
57
          network_interface {
             ipv4_address = var.ipaddr
58
             ipv4_netmask = 24
59
60
          ipv4_gateway = "gateway address here"
          dns_server_list = ["dns addresses here", "8.8.8.8"]
63
        }
     }
65
```

Picture 12. Main.tf shows basic VMware terraform provider setup for template use.



Picture 13. Provider.tf that is used to connect to vCenter.

```
1 vsphere_user= "administrator@vcenter.local"
2 vsphere_password= "password here"
3 vsphere_server= "vcenteraddresshere"
Picture 14. Terraform.tfvars for vCenter login.
```



Picture 15. Vars.tf for variables.

Before we can use the Terraform, we still need to create the Linux template that we will use as seen on picture twelve.

4.2.1 Operating System and Templates

We are using Red Hat Enterprise Linux 8.6 as the operating system of choice for our virtual machines that work as application servers. We install Red Hat normally on the vCenter. Then we register it on our Red Hat account so we can download updates and other necessary packages. You can do this with command sudo subscription-manager register --username [username] --auto-attach. This registers the Red Hat without showing the password on your command line history. Use the sudo subscription-manager list to see if the virtual machine is registered. After updating, we need to install perl and vm-open-tools to make sure Terraform works on the template. This can be achieved using simple command sudo dnf install open-vm-tools perl. Reboot the virtual machine and remove the subscription with sudo subscription-manager unregister to also remove the NICs from it. NICs are removed so the template does not cause MAC problems when cloned. Turn VM off and press convert to template.

This template will be used by Terraform to clone new virtual machines when the need arises. Templates are also easy to convert back to VMs if you want to keep the template up to date. Updating the template regularly keeps your automation environment faster when Ansible playbook does not need to check and download updates every time you run it.

4.2.2 Terraform Run and Ansible Integration

Normally you would go to the Terraform directory on your WSL and run terraform init to check that the plan works. Then terraform plan to insert needed items like hostname. After that, terraform apply to apply the plan and write yes on the question. Instead, we have automated this with bash script. We can run this script on Linux using sh [script name] [variables] etc. This clones the new virtual machine out of the template we made earlier and creates a directory of it on our local WSL to manage later. We move to this directory and use terraform show to see our configuration on the virtual machine we just made. You can also delete the VM from this directory given you have connection to the vCenter using terraform destroy. Creating VMs from templates takes about 1-3 minutes but deleting it with destroy is almost instant and permanent.

Now we can implement the bash script to the Ansible playbook by making it into a playbook role called terraform and making a simple main.yml file in the role/terraform/tasks directory as in picture sixteen.



Picture 16. Simple Ansible tasks that runs the local script on the WSL.

4.3 Ansible Roles Configuration

Now that Ansible can call the Terraform script with the role we just made, we can focus on how to automate the Red Hat registration part. This can also be done with role task in our playbook. Roles are the basis on how this playbook works in practice. Ansible goes through them from top to bottom one at a time. The playbook will prompt variables as questions for the roles as seen in picture seventeen below.

```
2 #Run Terraform script
    name: Run terraform and ssh scripts locally
3 -
    hosts: localhost
 6
    roles:
      - terraform
    vars_prompt:

    name: vmname #variable

10
         private: no
         prompt: "Write VM's name for terraform script"
12
       - name: ipaddr #variable
14
        private: no
         prompt: "Write VM's IP address for terraform script"
17 #Roles for rhel and extra ssh user shell
18 - name: Active rhel, create passwordless ssh user and update
    hosts: standalone hosts
    gather_facts: yes
20
    remote user: root
21
    vars:
23
       ansible_ssh_pass: "{{ ssh_passwd }}"
    become: yes
24
    become method: sudo
26
27
    roles:
28
      - rhelsub
29
       - ssh
      - updaterhel
30
    vars prompt:
32
33
      - name: ssh_passwd #variable
34
         private: yes
        prompt: "Enter target's root password"
      - name: org_activationkey #variable
         private: no
         prompt: "Enter Redhat activation key name"
       - name: org id #variable
39
         private: no
         prompt: "Enter Redhat Organization ID"
41
```

Picture 17. Idea how roles and vars_prompt work in playbooks.

Roles take these prompts as variables during the playbook run and apply them to the right places. As seen in pictures 16 and 17, variables are marked as

"{{ variable }}" in Ansible syntax. Red Hat registration role is made with a simple main.yml file in roles/rhelsub/tasks. We use variables and the Red Hat activation key to automate the task.

1	
2	# Subscription for rhel
3	 name: Register with activation key
4	community.general.redhat_subscription:
5	state: present
6	<pre>activationkey: "{{ org_activationkey }}"</pre>
7	org_id: "{{

Picture 18. Red Hat subscription with variables.

SSH role will create the passwordless SSH user we need for the last part of the playbook that is not shown in picture seventeen. We will not get into this because it is a production user at Cimcorp, but it simply creates the user at the target VM and gives it the SSH keys from a similarly named user on our WSL. This way, using this user on our WSL and having the same user at target makes running the production Ansible roles easy.

The last role we created for the playbook is simple updaterhel. This will check and update the target for the latest packages. As we can see, it is easy to add more modules to playbook as roles.



Picture 19. Updates target Red Hat VM.

4.3.1 Playbook Structure

So far, we have seen some parts of Ansible playbooks but we will go through the bigger picture now. At its core, Ansible playbook can be as simple as just one playbook.yml and ansible.cfg file in the same directory that has the line:

[defaults]

inventory=inventory

This will direct the default inventory hosts location to our current directory. Then we need the inventory directory that has the hosts file. We need to edit the hosts file to connect to desired target VMs. Lastly, the roles directory for all the different roles we have for playbook such as terraform and rhelsub as mentioned earlier.

4.3.2 Playbook Run

Before we can run our playbook, we still need to make sure the inventory/hosts file has the right hostname and IP address line for the hosts part in the playbook. When this is done, we can finally run our playbook on the vCenter. This happens with a simple command on WSL using ansible-playbook playbookname.yml. This will start the playbook by first running Terraform locally on the WSL. It prompts the user for hostname and IP address.

When the Terraform role is done, it will connect remotely to the new virtual machine as the root user. This will prompt the user for the target root password and Red Hat activation key name and organization ID. It will register the VM on the Red Hat account and create the SSH user on the target. Lastly, check for updates in case the template is out of date.

Playbook will continue to create the application server. I will not be able to show this because it is part of the current production at Cimcorp, but this part is done in the same way as the other roles but more complicated modules in the tasks files. All the guidance to create complicated Ansible playbooks can be found on their official documentation site.

4.4 Results

When the playbook has ran its course, we can see the results at the end. (Picture 20) The Red Hat virtual machine now has the needed applications running on it so that we can continue remotely and activate applications giving out needed commands. We can connect to this VM using SSH on our WSL or using the remote console on vCenter. Automating some parts like the creation of VMs saved us time and effort that can be used for other more important work.



Picture 20. Playbook was successful despite a few skipped and unreachable tasks.

5 CONCLUSIONS

In this thesis I looked into how to better automate the creation of application servers at Cimcorp. The results were that the playbook worked but left more questions than answers. Many things could be improved in the creations of the virtual machines. Automation is a never-ending task and things could always be simplified for less likelihood of human errors. Real improvements that could be done to this work are how to connect to the target host while not using root user and improving the Terraform and Ansible code. Streamlining to user experience to get rid of the manual parts like changing the host file. The Terraform script I used for the automation could also be improved and expanded. The problem with creating and using the template on the server that goes to the customer needs to be solved too. The solution for all these is more time and experience with the code and scripts. Unfortunately, in a production environment, extra time to do development work is extremely limited.

REFERENCES

Ansible, & Red Hat. (2022). *How Ansible works*. Ansible is Simple IT Automation. Retrieved August 8, 2022, from <u>https://www.ansible.com/overview/how-ansible-works</u>

Ansible. (2022, August 5). *How to build your inventory* — *Ansible documentation*. Ansible Documentation. Retrieved August 9, 2022, from https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

Ansible. (2022, May 27). *Introduction to modules — Ansible documentation*. Ansible Documentation. Retrieved August 10, 2022, from <u>https://docs.ansible.com/ansible/2.9/user_guide/modules_intro.html</u>

Ansible. (2022, August 7). YAML syntax — Ansible documentation. Ansible Documentation. Retrieved August 8, 2022, from https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

Ansible. (2022, August 5). *Intro to playbooks — Ansible documentation*. Ansible Documentation. Retrieved August 9, 2022, from https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

Carey, S. (2021). What is infrastructure as code? Automating your infrastructure builds. *InfoWorld.Com*

HashiCorp. (2022). *Providers - Configuration language | Terraform by HashiCorp*. Terraform by HashiCorp. Retrieved August 9, 2022, from <u>https://www.terraform.io/language/providers</u>

HashiCorp. (2022). *Syntax - Configuration language | Terraform by HashiCorp.* Terraform by HashiCorp. Retrieved August 11, 2022, from <u>https://www.terraform.io/language/syntax/configuration</u> HashiCorp. (2022). *Files and directories - Configuration language | Terraform by HashiCorp*. Terraform by HashiCorp. Retrieved August 15, 2022, from https://www.terraform.io/language/files

HashiCorp. (2022). *Overview - Configuration language | Terraform by HashiCorp.* Terraform by HashiCorp. Retrieved August 15, 2022, from <u>https://www.terraform.io/language</u>

HashiCorp. (2022). *Types and values - Configuration language | Terraform by HashiCorp*. Terraform by HashiCorp. Retrieved August 15, 2022, from https://www.terraform.io/language/expressions/types

Howard, M. (2022). Terraform -- Automating Infrastructure as a Service.

IBM. (2019, June 19). *Virtualization-a-complete-guide*. Retrieved August 18, 2022, from <u>https://www.ibm.com/cloud/learn/virtualization-a-complete-guide</u>

Ingalls, S. (2021, November 5). *What is an application server?* ServerWatch. Retrieved August 23, 2022, from <u>https://www.serverwatch.com/guides/application-server/</u>

Marshall, N., Brown, M., Fritz, G. B., & Johnson, R. (2018). *Mastering VMware vSphere 6.7.* John Wiley & Sons.

Oh, D., Freeman, J., & Locati, F. A. (2020). Practical Ansible 2: Automate infrastructure, manage configuration, and deploy applications with Ansible 2.9

Phipps, J. (2022, July 27). *Web servers vs application servers: What's the difference?* Woodsia. Retrieved August 23, 2022, from <u>https://www.webopedia.com/servers/web-server-vs-application-server/</u>

Red Hat. (2018, March 2). *What is virtualization?* Retrieved August 18, 2022, from <u>https://www.redhat.com/en/topics/virtualization/what-is-virtualization</u>

Rubens, P. (2017). Why Ansible has become the devops darling for software automation. Cio, Retrieved August 7, 2022, from https://www.proquest.com/trade-journals/why-ansible-has-become-devops-darling-software/docview/1902032777/se-2

Seaton, J. (2011, August 10). *What is VMware ESXi server? - Definition from Techopedia*. Techopedia.com. Retrieved August 20, 2022, from <u>https://www.techopedia.com/definition/25979/vmware-esxi-server</u>

Smith, S. R. & Membrey, P. (2022). Beginning Ansible Concepts and Application: Provisioning, Configuring, and Managing Servers, Applications, and Their Dependencies. Apress L. P.

Strickland, J. (2008, June 2). *How server virtualization works*. HowStuffWorks. Retrieved August 18, 2022, from <u>https://computer.howstuffworks.com/server-virtualization.htm</u>

Verona, J. (2016). Practical DevOps

VMware. (2022). VMware vSphere documentation. VMware Docs Home. Retrieved August 20, 2022, from <u>https://docs.vmware.com/en/VMware-vSphere/index.html</u>

VMware. (2022, August 10). *What is infrastructure automation? / VMware glossary*. Retrieved August 21, 2022, from https://www.vmware.com/topics/glossary/content/infrastructure-automation.html