



# **Ansible and Jenkins based solution for managing Virtual Machine pools.**

Anmol Arora

BACHELOR'S THESIS  
December 2024

Software Engineering

## **ABSTRACT**

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Bachelor's Degree Programme in Software Engineering

**ANMOL ARORA:**

Ansible and Jenkins based solution for managing Virtual Machine pools.

Bachelor's thesis 38 pages

April 2024

---

This thesis is dedicated to creating an Ansible and Jenkins based solution for managing and deploying virtual machines across the global development ecosystem of a major enterprise specializing in advanced cargo movement and material handling. The motivation behind this initiative arises from the developers' need for a more efficient method of obtaining updated VMs. The primary goal is to establish a universally accessible framework for crafting VM templates.

The thesis serves as a comprehensive guide, detailing the setup and configuration of the environment and displaying the use of Ansible playbooks to accurately instantiate virtual machines. It provides a step-by-step walkthrough for developing and implementing the proposed solution. By the conclusion of this thesis, readers will gain a profound understanding of the principles, methodologies, and practical applications involved in automating VM template customization.

The approach combines the versatility of Ansible with a user-friendly framework, enabling developers to easily adapt and employ the solution across diverse scenarios within the enterprise. This not only addresses the immediate need for updated VMs but also contributes to a more streamlined and accessible process for all members of the corporate environment.<sup>1</sup>

---

<sup>1</sup> Key words: Ansible-based solution, virtual machines, VM templates, automating, playbooks.

## CONTENTS

1 INTRODUCTION .....	6
1.2 Motivation for the project .....	6
2 BACKGROUND .....	7
2.1 Ansible.....	7
2.1.1 Ansible ad-hoc commands.....	8
2.1.2 Ansible patterns and modules.....	9
2.1.3 Ansible Roles.....	10
2.1.4 Ansible Playbooks.....	12
2.1.5 Ansible Tags .....	14
2.2 Jenkins .....	15
2.2.1 Jenkins Plugins .....	15
3 EXECUTION .....	17
3.1 System architecture.....	17
3.1.1 Overview diagram and introduction .....	17
3.1.2 Main components .....	17
3.1.3 Project Directory structure .....	18
3.2 Client Side .....	19
3.2.1 Jenkins Job Parameters .....	20
3.2.2 Jenkins Working .....	21
3.2.3 Pipeline deep dive .....	23
3.3 Version Control.....	25
3.4 Jenkins Agent.....	26
3.4.1 Docker File.....	27
3.4.2 Ansible .....	28
3.5 Server Side.....	31
3.5.1 XenServer CLI .....	31
4 RESULTS AND DISUCSSION.....	33

4.1 Old VS New .....	33
4.2 Future plans and development .....	35
5 CONCLUSION .....	36
REFERENCES .....	37

## ABBREVIATIONS

CLI	Command Line Interface
GROOVY	Scripting language used to define Jenkins Pipelines
IT	Information Technology
INI	Initialisation
IP	Internet Protocol
JINJA	Web template engine for Python
JRE	Java Runtime Environment
JSON	JavaScript Object Notion
NODES	Target device managed by Ansible.
SCM	Source Code Management
SSH	Secure Shell Server
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
XE	XenServer Edition
YAML	YAML Ain't Markup Language
XVA	X-Value Adjustment / virtual appliance format

## **1 INTRODUCTION**

The thesis focuses on leveraging Ansible and Jenkins to address the challenges in managing and distributing virtual machines within a multinational company specialising in advanced cargo and material handling. The current manual and outdated distribution process consumes valuable man-hours, prompting the need for automation to enhance efficiency.

The goal is to create a streamlined solution that empowers developers to easily customise VM templates, fostering a more flexible and agile development workflow. This aligns with the company's broader objective of promoting automation.

The proposed solution, utilising Ansible, is chosen for its declarative configuration, agentless architecture, and Infrastructure as Code (IaC) approach. The design emphasises simplicity, enabling easy setup and configuration adaptability for various environments.

The completion criteria for the thesis involve developers using Jenkins pipelines alongside Ansible playbooks to autonomously create personalised virtual machines. These virtual machines are used within the company environment for various purposes like developer VMs and for running company internal software for testing and further development. Extensive documentation will accompany the solution to ensure comprehensive understanding and implementation.

By adopting Ansible's capabilities and integrating it into the Jenkins development pipeline, the thesis aims to not only resolve the existing inefficiencies but also contribute to a more dynamic and responsive adaptation of virtual environments to evolving project requirements.

### **1.2 Motivation for the project**

The project started as a suggestion ticket in Jira which was assigned to the DevOps team, the suggestion was to create a faster way to distribute virtual

machine template images in the development network consisting of 20+ server pools and a total of 60+ servers.

After some internal discussions, work on this ticket was started. The primary goal was to use Ansible to create several playbooks which would work in conjunction with the already existing xen-utilities installed in all the servers. In addition, for easier usage and accessibility, it was decided to set up a Jenkins pipeline as well. Finally, the project should also allow distribution of the newly created templates to all the pools, additionally customisation such as name and snapshot name for the VM can also be set.

## **2 BACKGROUND**

This chapter will explore the theoretical foundation, benefits, and drawbacks of the primary technologies used in this project. Details on usage and application will be discussed and presented in the following sections.

### **2.1 Ansible**

Ansible is a Python-based open-source tool which operates via command-line for automating IT tasks. It can configure systems, deploy software, and orchestrate complex workflows.

Simplicity and ease of use are the main strengths of Ansible. It features minimal moving parts and has a strong focus on security and reliability. It uses OpenSSH for transport and uses a human readable language that does not require a lot of training.

Originally written by Michael DeHaan in 2012, and acquired by Red Hat in 2015, Ansible can be used to configure both Unix-like systems and Microsoft Windows. Ansible is agentless and utilizes temporary remote connection (SSH) or Windows Remote Management (WinRM) via PowerShell execution.

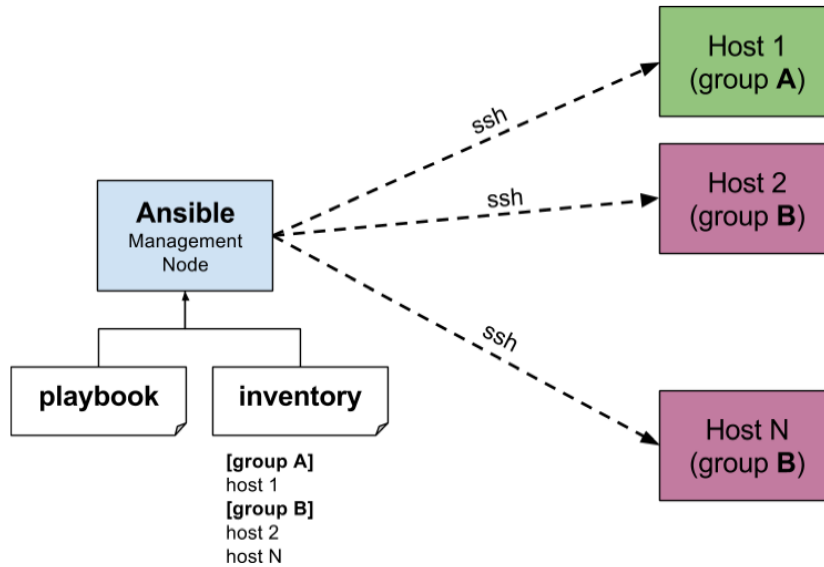


Figure 1: [Ansible basic](#) architecture (Rajesh Kumar, July 4,2019).

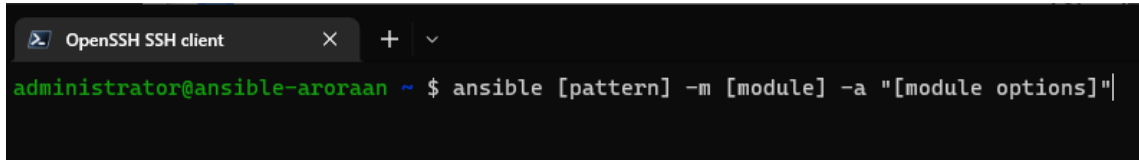
Some of the design goals of Ansible are:

- **Security:** Ansible operates without deploying agents to nodes, relying solely on OpenSSH and Python on managed nodes.
- **Reliability:** With careful scripting, Ansible playbooks can be made idempotent to prevent unexpected impacts on managed systems, although non-idempotent playbooks are also possible.
- **Consistency:** Ansible enables the creation of consistent environments effortlessly.
- **Easy learning curve:** Playbooks employ a straightforward and descriptive language based on YAML and Jinja templates.
- **Minimalistic:** Management systems should avoid adding unnecessary dependencies to the environment.

### 2.1.1 Ansible ad-hoc commands

When starting with ansible it is often recommended to get familiar with some ad-hoc command line commands. Ansible ad hoc commands utilize the `/usr/bin/ansible` command-line tool to automate tasks across one or multiple managed nodes. Although convenient and swift, ad hoc commands lack reusability.





```

OpenSSH SSH client
administrator@ansible-aroraan ~ $ ansible [pattern] -m [module] -a "[module options]"

```

Figure 2: Basic structure of ad-hoc in ansible.

The '-a' option in Ansible accepts options either in the 'key=value' syntax or as a JSON string enclosed within '{' and '}' for handling more complex option structures.

### 2.1.2 Ansible patterns and modules

With the help of Ansible Patterns, one can run commands and playbooks against specific nodes and/or groups in the inventory. Ansible pattern can refer to a single host, an IP address, an inventory group, a set of groups, or all hosts in the inventory. Patterns in Ansible are extremely flexible, allowing for easy exclusion or inclusion of subsets of hosts. Additionally, they support the use of wildcards and regular expressions, further enhancing their versatility.

Table 1: Common patterns for specifying hosts and groups in inventory.

Description	Pattern(s)	Targets
All hosts	all (or *)	
Singular host	Node1	
Multiple hosts	node1:node2(or node1;node2)	
Single group	database	
Multiple groups	database:application	all hosts in database plus all hosts in application
Excluding groups	database:!georgia	all hosts in database except those in georgia
Combining groups	database:&staging	all hosts in database that are also in staging

Modules, also known as "task plugins" or "library plugins," are independent units of code that can be utilized either from the command line or within playbook tasks. Ansible executes each module, typically on the remote target node, and gathers return values accordingly.

Module can be executed directly from the command line:

```
ansible webservers -m service -a "name=httpd state=started"
ansible webservers -m ping
ansible webservers -m command -a "/sbin/reboot -t now"
```

Figure 3: Ad-hoc command showing usage of modules.

All modules return JSON format data. When used within an Ansible playbook, modules can trigger 'change events,' which can lead to execution of 'handlers' to perform additional tasks.

Command line can also be used to access documentation for any module.

```
ansible-doc yum
```

Figure 4: Command line tool to access ansible documentation.

### 2.1.3 Ansible Roles

Ansible Roles offer a structured framework for organizing tasks, variables, handlers, metadata, templates, and other files. They facilitate code reuse and sharing in Ansible. With Roles, Ansible code can be referenced and reused in playbooks with minimal effort, often just requiring a few lines of code.

The directory structure of an Ansible role has eight main standard directories. At least one of these directories must be included, others can be omitted if the role does not require them.



Figure 5: Ansible project structure with roles folder.

Usually, a role has the tasks and default directory in it, the other directories are for specific situations. All eight of them are:

- defaults: Holds default values for the role's variables.
- vars: Contains variables specifically defined for the role.
- tasks: Consists of a list of tasks to be executed by the role, similar to the task section of a playbook.
- files: Stores static and custom files utilized by the role for various tasks.
- templates: Stores Jinja2 template files utilized by tasks within the role, enabling dynamic expressions and variable access.
- handlers: Contains handlers that check for specific changes on the machine and run only if those changes occur, such as restarting a service after an update.
- meta: Includes metadata information for the role, including dependencies and author details.
- tests: Holds configuration files related to role testing.

Within these directories the actual name of the YAML file stays the same. It is named as 'main.yaml'.

## 2.1.4 Ansible Playbooks

Playbooks serve as automation blueprints in YAML format that Ansible utilizes to deploy and configure managed hosts.

The crucial concepts of Ansible are the following:

- **Playbook:** A collection of plays that govern the sequence in which Ansible executes operations, from start to finish, to accomplish an operation.
- **Play:** A sequential list of tasks assigned to managed nodes within an inventory.
- **Task:** A command to a single module outlining the actions that Ansible executes.
- **Module:** A discrete piece of code or binary executed by Ansible on managed nodes.

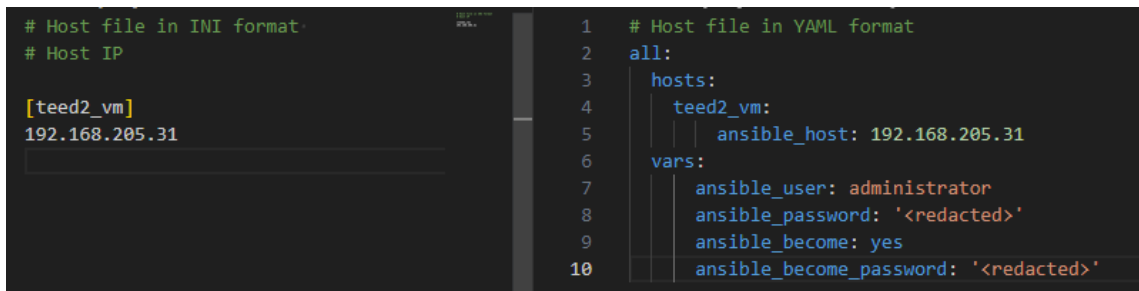
```
---
- name: My first play
  hosts: teed2_vm
  vars:
    ansible_user: 'administrator'
    ansible_password: '<redacted>'
    ansible_become: yes
    ansible_become_password: '<redacted>'
  tasks:
    - name: Just a ping
      ping:
        register: ping_result
    - name: Show ping output
      debug:
        var: ping_result
```

Figure 6: First playbook using module ping.

In the above mock playbook, the basic structure of an Ansible playbook can be noticed, starting with the name of the playbook which the user can set as per their needs, usually an indication to what the playbook will try to achieve. Then the hosts are defined, the host name given should correspond to one of the entries

in the inventory file as well. Inventory files are discussed more in detail shortly. After hosts there is a possibility to set some playbook level variables, the scope of these variables is global and are applied to all the tasks/roles in the playbook unless overwritten. Lastly in this playbook, tasks are described. There are two simple tasks, the first one is ping which uses the inbuilt ping module in Ansible and checks the connection to the host. The next one is the debug module to show the output of the ping.

Since Ansible communicates via SSH, passing credentials is important and vars are the easiest way to do that. The inventory/hosts file should also be structured to include IP addresses of the target machines. The inventory file can be written in either YAML format like the playbook or INI format.



```
# Host file in INI format
# Host IP

[teed2_vm]
192.168.205.31

1 # Host file in YAML format
2 all:
3   hosts:
4     teed2_vm:
5       ansible_host: 192.168.205.31
6   vars:
7     ansible_user: administrator
8     ansible_password: '<redacted>'
9     ansible_become: yes
10    ansible_become_password: '<redacted>'
```

Figure 7: Comparison between INI and YAML syntax.

In the YAML syntax variables can be added under the host section which makes the playbook look cleaner.

The syntax for running playbooks is:

```
'$ ansible-playbook -i <hostfile> <playbook name>'
```

```

administrator@ansible-aroraan ~/myProject $ ansible-playbook -i inventory test-playbook.yaml

PLAY [My first play] *****

TASK [Gathering Facts] *****
ok: [teed2_vm]

TASK [Just a ping] *****
ok: [teed2_vm]

TASK [Show ping output] *****
ok: [teed2_vm] => {
  "ping_result": {
    "changed": false,
    "failed": false,
    "ping": "pong"
  }
}

PLAY RECAP *****
teed2_vm : ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0

```

Figure 8: Running the playbook.

Ansible returns information about the playbook run in JSON format and the result from the ping module is returned as 'pong' if successful.

### 2.1.5 Ansible Tags

In Ansible, tags are a powerful mechanism used to selectively execute specific tasks within a playbook based on user-defined labels. Tags allow users to categorize tasks and control their execution during playbook runs. When defining tasks within a playbook, users can assign one or more tags to each task using the **tags** parameter. These tags serve as identifiers that can be used to include or exclude tasks from execution.

During playbook execution, users can specify which tasks to run by specifying tags on the command line. Ansible provides options such as **--tags** to specify which tags to include and **--skip-tags** to specify which tags to exclude. This allows for fine-grained control over playbook execution, enabling users to target specific tasks based on their tags.

Additionally, Ansible provides an inbuilt variable called **ansible\_run\_tags**, which contains all the tags specified on the command line when the playbook is executed. This variable can be accessed within the playbook to dynamically adjust task execution based on the tags provided at runtime.

## 2.2 Jenkins

Jenkins is a self-contained, open-source automation server utilized for automating various tasks associated with building, testing, and delivering software.

It can be installed via native system packages, Docker, or as a standalone application on any machine equipped with a Java Runtime Environment (JRE). Initially named Hudson, the project was rebranded as Jenkins in 2011 following a dispute with Oracle, which had forked the project and laid claim to the original name. Despite Oracle's continued development of the forked project under the name Hudson for a period, it has since become obsolete and is no longer maintained.

Jenkins offers multiple ways in which a build can be triggered:

- **Webhook:** This feature acts as a trigger mechanism that responds to commits pushed to a version control system, initiating actions or processes within the system.
- **Scheduling:** Similar to cron job, this functionality enables the automated execution of tasks or processes at predefined intervals or time schedules.
- **URL Endpoint:** Allows for the initiation of a build process through a specific URL, providing a straightforward and direct method for triggering builds externally.
- **Invoke:** Provides the capability for one build process to initiate or start another build, facilitating sequential or dependent build workflows.
- **Queue Management:** Manages the order and execution of builds within a queue, ensuring that builds are processed in a systematic manner, with consideration given to dependencies and resource availability.

### 2.2.1 Jenkins Plugins

Jenkins also supports plugins to add extra functionality and extends its use to projects. Plugins are available for integrating Jenkins with most version control

systems and bug databases. Plugins can also change the look and feel of Jenkins. Plugins that are in use for this project:

- Credentials Binding - Allows storing credentials in Jenkins and a layer of security so that credentials are not passed as clear text or shown up in pipeline logs.
- Docker pipeline – Allows building, testing, and using Docker images from Jenkins pipeline projects.
- GitLab plugin - Triggers builds in Jenkins when code is submitted, or merge requests is opened/updated.
- Plugin Usage - Plugin that gives possibility to analyse the usage of all installed plugins.



### 3 EXECUTION

This chapter aims to outline the primary components of both Ansible playbook and Jenkins pipeline, covering their design, development, operation, and how they interrelate.

#### 3.1 System architecture

##### 3.1.1 Overview diagram and introduction

The template update and distribution job is a Jenkins pipeline, the main components and how they interact with each other are depicted below in Figure 9. The system architecture revolves around a Jenkins job that serves as the central control point for managing parameters and configurations. Leveraging the declarative pipeline approach, the pipeline code is sourced from a GitLab repository, ensuring version-controlled and easily maintainable workflows.

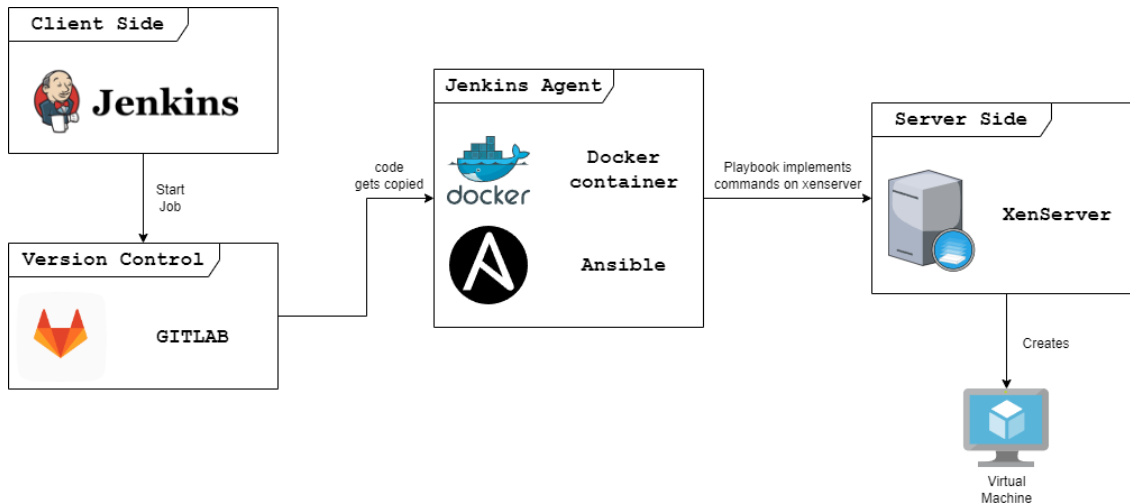


Figure 9: Overview diagram of the Ansible solution with other components.

##### 3.1.2 Main components

The main components, their purpose, and roles in the Ansible based solution are described in Table 2 below.

	<b>Components</b>	<b>Description</b>
<b>Client</b>	Jenkins	Automation server for executing pipeline workflows, triggered by events and parameters defined in GitLab
	GitLab	Version control repository storing pipeline code and configurations, facilitating version-controlled automation processes.
<b>Jenkins Agent</b>	Docker	Containerization platform providing isolated environments for Jenkins agents, ensuring consistent pipeline execution.
	Ansible	Configuration management tool for provisioning and managing virtual machines, orchestrating VM creation, software installation, templating, and distribution.
<b>Server Side</b>	XenServer CLI	Managed pool of virtual machines interacted with by Ansible for creating, configuring, templating, and distributing VMs across the development environment.
	Virtual Machine	Virtual machine created in XenServer pool where updates will be installed in next stages of Jenkins pipeline

Table 2: Main components used in the Ansible solution.

### 3.1.3 Project Directory structure

This is a cleaned-up version of the project directory and shows all the relevant files and folders.

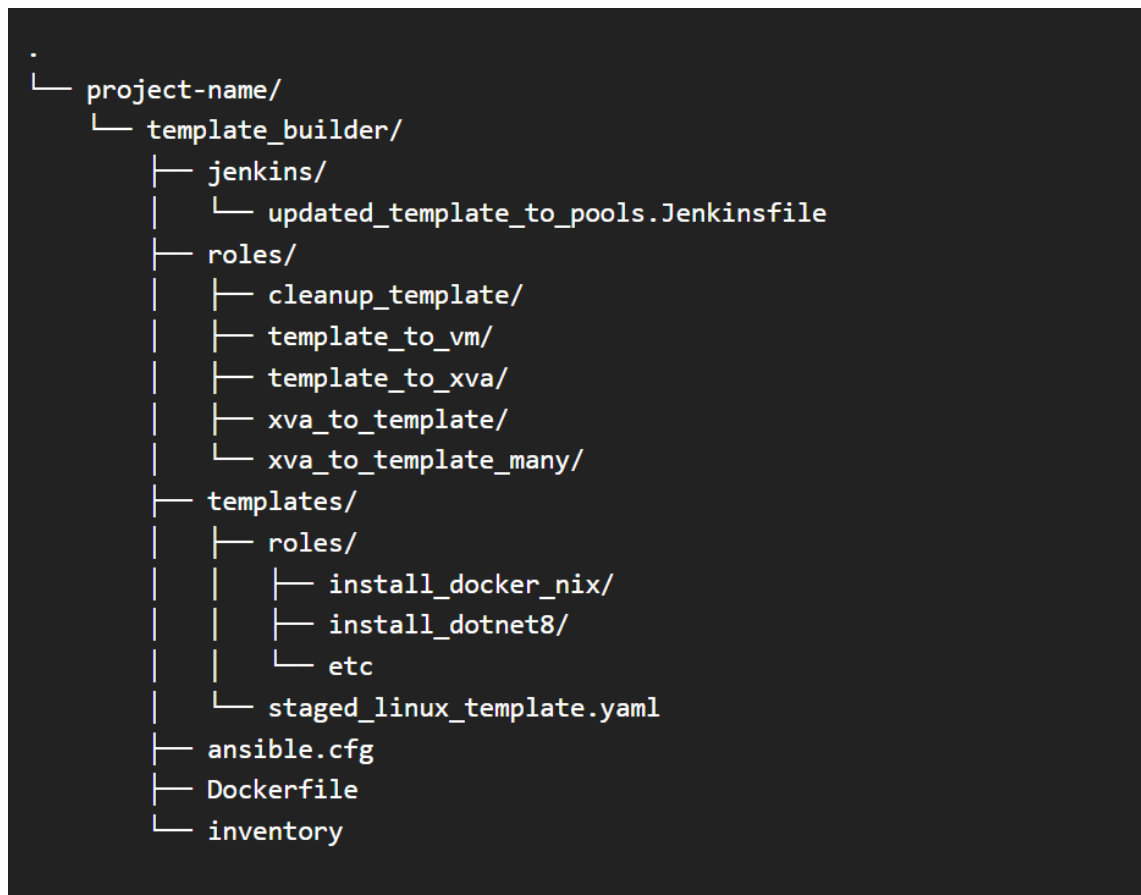


Figure 10: Project directory structure.

The Jenkins pipeline is outlined in the '*jenkins*' directory under the name '***updated\_template\_to\_pools.Jenkinsfile***'. In the '*roles*' directory, all essential roles for template creation and distribution are stored. Within the '*templates*' folder, there exists a sub-folder named '*roles*', housing all roles related to component installation. The primary playbook responsible for executing these roles, '***staged\_linux\_template.yaml***', is also located in the '*templates*' directory. Additionally, within the main directory, there are files pertinent to the Docker container, including the '*Dockerfile*' detailing container specifications and image usage. Furthermore, the '*ansible.cfg*' file is copied to the Docker container. Lastly, the '*inventory*' file specifies the hosts targeted by ansible commands.

### 3.2 Client Side

### 3.2.1 Jenkins Job Parameters

The client side is a single job on the Jenkins Web application where the following parameters can be set:

- Refresh – Ticking this option enables dry run of the pipeline and checks for updates. No ansible roles are ran.
- image\_name – Takes in the name of the template image, which is used to create base vm and install updates to it. Base image can either be stored in network share drive or in Artifactory.
- host\_ip – Takes in IP of the server pool in which the VM will be made.
- updated\_template\_name – Takes in name for the updated template that will be created at the end of the job.
- vm\_name – Takes in name of the virtual machine which is created from the base image and then updated.
- snapshot\_name – Takes in name for the snapshot of the virtual machine that is created.
- hostlist\_raw – Takes in IP addresses of all server pools where updated template must be copied to. Delimiter is comma “,”
- PoolDistribute – Ticking this option enables the pool distribution part of the pipeline.
- GIT\_BRANCH – Takes in from which branch to build pipeline.
- secret – Login credentials used by ansible roles stored in Jenkins using ‘Credentials’.

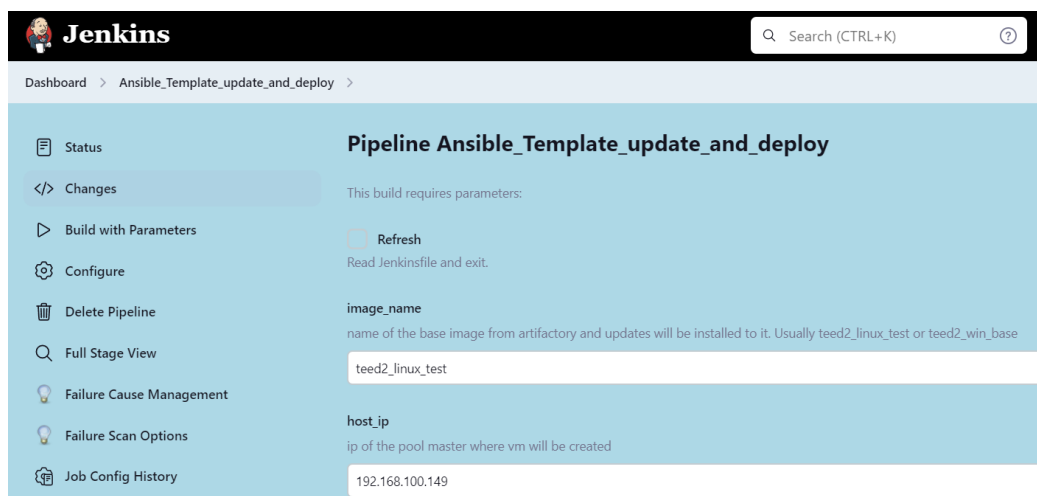
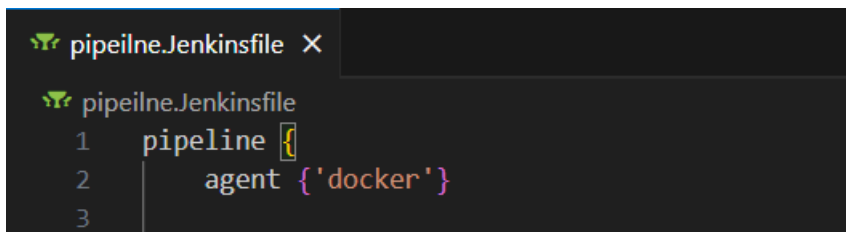


Figure 11: Jenkins Web UI

### 3.2.2 Jenkins Working

In Jenkins, Groovy is used to write pipeline scripts that define the job configuration and orchestrate the deployment process. While Groovy is an object-oriented programming language its usage in Jenkins is purely for scripting purposes. Jenkins Groovy scripts typically involve defining stages, steps, and conditional logic to control the execution flow of a job. This way of configuring a Jenkins job is referred to as Declarative Pipeline.

In Jenkins, the machine on which the jobs are running is called an 'agent.' The Jenkins agent is defined in the pipeline script which is written in Groovy and stored using GitLab version control. The agent can either be the Jenkins server itself or a Docker container within the server. Docker containers provide a consistent and isolated runtime environment, ensuring that each execution of the pipeline occurs in a controlled environment with the necessary dependencies and configurations, regardless of the underlying host system or server setup.



```

pipeline {
  agent { 'docker' }
}

```

Figure 12: Setting agent in groovy pipeline.

The parameters mentioned above in section '[3.2.1 Main Technologies](#)', which are needed for the execution of a job are also defined in the Groovy script, each parameter has its type and some extra variables like 'default value' or 'choices'.



```

parameters {
  booleanParam(name: 'Refresh', defaultValue: false, description: 'Read Jenkinsfile and e
  string(name: 'image_name', defaultValue: "teed2_linux_test", description: "name of the
  string(name: 'host_ip', defaultValue: "192.168.100.149", description: "ip of the pool m
  string(name: 'template_name', defaultValue: "teed2_linux_test", description: "Name of t
  string(name: 'updated_template_name', defaultValue: "updated_ansible_template", descrip
  string(name: 'vm_name', defaultValue: "vm_by_ansible_template", description: "Name for
  string(name: 'snapshot_name', defaultValue: "Ansible_template_snap_01", description: "N
  text(name: 'template_description', description: "GIT branch and hash gets added here")
}

```

Figure 13: Parameter types in Groovy.

The rest of the Groovy script describes the stages of the pipeline and runs the individual commands in those stages. A 'stage' in Jenkins is the primary building block in a pipeline. It utilizes the 'Pipeline Stage Setup' plugin and requires wrapping all steps within the defined stage. This makes the boundaries of each stage obvious and predictable.

```
stages {  
    stage("Read jenkins file and exit"){  
        when {  
            expression { params.Refresh == true }  
        }  
        steps {  
            echo("Ended pipeline early.")  
        }  
    }  
    stage("Run Jenkins File check"){ ...  
}
```

Figure 14: Pipeline stages.

Multiple 'stage' can be put together in a stages section to make the code more readable as well. The main stage is the 'Run Jenkins File check' which has another subsequent 'stages' section. The Jenkins UI shows all the stages at the jobs home page and their status.



Figure 15: Stage view in Jenkins UI

### 3.2.3 Pipeline deep dive

This section will focus on the 'Run Jenkins File check' stage of the pipeline and breakdown the structure of the stage in more detail.

```

stage("Run Jenkins File check"){
  agent {
    dockerfile {
      dir 'template_builder/'
      reuseNode true
    }
  }
}

```

Figure 16: Agent reads docker file location.

At the start of the pipeline, 'docker' was defined as the choice for Jenkins agent, in the stage 'Run Jenkins File check' the location for the Dockerfile to build the docker container from is provided. At the start of a job which uses this stage, the docker container is built and then all commands are run from within this docker container.

The next stage is 'Generating template description,' all templates that are created and stored in server pools need to have some sort of description so that the components installed can be traced. For this reason, git commit hash is used as the template description as git commit hashes are unique.

```

stage("Generating template description"){
  when {
    expression { params.Refresh == false }
  }
  steps{
    script{
      def descriptionValue = "Branch Name: ${env.GIT_BRANCH}; GIT hash: ${env.GIT_COMMIT}"
      withEnv(["template_description=${descriptionValue}"]){
        echo "Template Description: '$template_description'"
      }
    }
  }
}

```

Figure 17: Generating template description.

The next two stages are the ones responsible for running the ansible playbooks for the creation of the virtual machine and the components installation.

```

// define stages here for steps in vm creation and distribution
stage("Make updated vm and install components") {
  steps {
    withCredentials([usernamePassword(credentialsId: 'artifactory_secret', passwordVariable: 'PASSWORD', usernameVariable: 'USERNAME')]) {
      sh """ansible-playbook -i template_builder/inventory template_builder/templates/${params.ansible_playbook_name}
      --tags "template_to_vm" --tags "installer" -e vm_name=${params.vm_name} -e updated_template_name=${params.updated_template_name}
      -e template_name=${params.template_name} -e ansible_pass=${params.secret} -e image_filename=${params.image_name}
      -e artifactory_pass=${PASSWORD} -e host=${params.host_ip}"""
    }
  }
}

stage("Make template from newly created VM and upload to artifactory") {
  steps {
    script{
      def descriptionValue = "${env.GIT_COMMIT}"
      withEnv(["template_description=${descriptionValue}"]){
        withCredentials([usernamePassword(credentialsId: 'artifactory_secret', passwordVariable: 'PASSWORD', usernameVariable: 'USERNAME')]) {
          sh """ansible-playbook -i template_builder/inventory template_builder/templates/${params.ansible_playbook_name}
          --tags "template_to_xva" -e vm_name=${params.vm_name} -e updated_template_name=${params.updated_template_name}
          -e template_name=${params.template_name} -e ansible_pass=${params.secret} -e image_filename=${params.image_name}
          -e artifactory_pass=${PASSWORD} -e host=${params.host_ip} -e template_description=${template_description}"""
        }
      }
    }
  }
}

```

Figure 18: Stages for template creation.

These stages use the ansible playbooks stored in a different folder in the repo and pass the parameters that were set at the start of the job via the Jenkins web UI to ansible.

The last stage of the pipeline is the template distribution stage, it is only triggered if the [PoolDistribute](#) option is ticked and enables template distribution to all specified server pools.



```

stage("Distribute template to pools"){
  when {
    expression { params.PoolDistribute == true }
  }
  steps {
    withCredentials([usernamePassword(credentialsId: 'artifactory_secret', passwordVariable: 'PASSWORD', usernameVariable: 'USERNAME')]){
      sh """ansible-playbook -i template_builder/inventory
      template_builder/templates/${params.ansible_playbook_name}
      --tags "xva_to_template_many"
      -e ansible_pass=${params.secret} -e image_filename=${params.updated_template_name}
      -e artifactory_pass=${PASSWORD} -e hostlist_raw=${params.hostlist_raw} -e host=${params.host_ip}"""
    }
  }
}

```

Figure 19: Template distribution stage.

Jenkins also gives the ability to define sections that should run at the end of a pipeline and depending on the outcome being 'success' or 'failure' the actions taken can be different.

```

post{
  failure {
    script {
      try {
        withCredentials([usernamePassword(credentialsId: 'arti
        sh """ansible-playbook -i template_builder/inventory t
      }
    }
    catch (Exception e) {
      echo 'Exception occurred: ' + e.toString()
      echo 'Check that environment was properly destroyed'
    }
    cleanWs(cleanWhenNotBuilt: false,
    deleteDirs: true,
    disableDeferredWipeout: true,
    notFailBuild: false)
  }
}

```

Figure 20: Jenkins post action.

In this pipeline in case of a failure a cleanup role is ran which deletes any virtual machine or template that might have been created during the process. The role is wrapped in a try-catch block and then parameters for workspace cleanup are set as well.

### 3.3 Version Control

The project uses GitLab for source code management and version control. The connection from GitLab to Jenkins is made from within the job configuration in Jenkins.

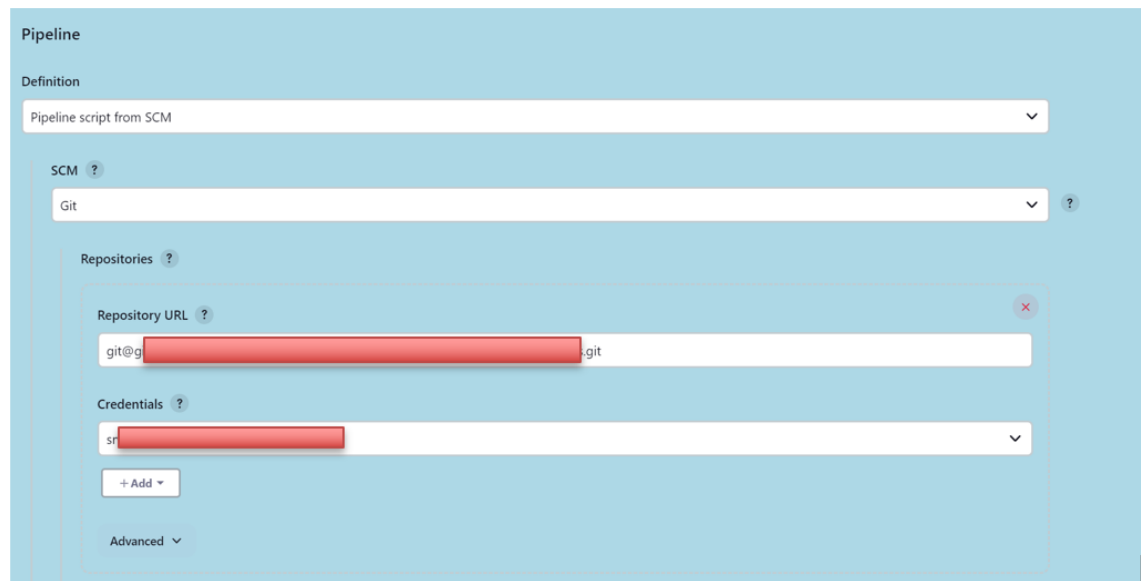


Figure 21: Jenkins SCM configurations.

Other options like what branch to build from and the path to Jenkinsfile are also set here.

### 3.4 Jenkins Agent

The Jenkins agent is an UbuntuServer20.04 virtual machine inside which docker is used to run the Jenkins job. In Jenkins, the agents are also referred to as 'Nodes' and can be viewed from the *Dashboard* → *Manage Jenkins* → *Nodes*.

The screenshot shows the Jenkins 'Nodes' page with a table of agent nodes. The table has the following columns: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The nodes listed are:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	ansible_builder	Linux (amd64)	In sync	1.17 GB	1.93 GB	1.17 GB	24ms
	Built-In Node	Linux (amd64)	In sync	384.75 GB	0 B	384.75 GB	0ms
	dockerworker1		N/A	N/A	N/A	N/A	N/A
	thesisdockerworker	Linux (amd64)	In sync	1.07 GB	1.92 GB	1.07 GB	23ms
Data obtained		8 min 34 sec	8 min 34 sec	8 min 34 sec	8 min 34 sec	8 min 34 sec	8 min 34 sec

Figure 22: Jenkins nodes.

The node for this job is called '**thesisdockerworker**,' names are customisable through node management. Since Jenkins is built on Java the connection from Jenkins to the node is also made via java .jar file.

The steps to setting up a node are quite easy:

- create new virtual machine from template UbuntuServer20.04 unless there are some other specific needs.
- Install Java, UbuntuServer20.04 template already has it installed.
- Download the Java .jar file from Jenkins.
- Create a Jenkins service to start/stop the .jar file.
- Configure the service to include your secret which is created by Jenkins to identify node.

### 3.4.1 Docker File

The docker image used for the Jenkins agent is ubuntu:20.04 as well, the container is built from the Dockerfile stored in the GitLab repository.

```

1  FROM ubuntu:20.04 AS deploy_base
2
3  RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y python3
4  && rm -rf /var/lib/apt/lists/*
5
6  RUN python3 -m pip install ansible==4.0.0 netaddr==0.9.0 pywinrm==0.4.3
7
8  WORKDIR /etc/ansible
9
10 COPY ansible.cfg .
11
12 WORKDIR /opt
13
14 CMD ["/bin/bash"]
15

```

Figure 23: Jenkins agent Dockerfile.

There are some packages defined in the Dockerfile as well which are needed by the container since docker images are light weight and do not come with any extra packages pre-installed. Once the image is built it can be reused by Jenkins and the job logs look like this.

```

[Pipeline] sh
+ docker build -t f2f2508fbdcf9366289297fec219e6f8c9d5dd1 -f template_builder//Dockerfile template_builder/
Sending build context to Docker daemon 141.3kB

Step 1/7 : FROM ubuntu:20.04 AS deploy_base
--> 18ca3f4297e7
Step 2/7 : RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y python3 python3-pip unixodbc-dev sshpass curl z
/var/lib/apt/lists/*
--> Using cache
--> eb58f6ca6195
Step 3/7 : RUN python3 -m pip install ansible==4.0.0 netaddr==0.9.0 pywinrm==0.4.3
--> Using cache
--> 153d9d892597
Step 4/7 : WORKDIR /etc/ansible
--> Using cache
--> 9756e02a7c7e
Step 5/7 : COPY ansible.cfg .
--> Using cache
--> 0565f22be8f3
Step 6/7 : WORKDIR /opt
--> Using cache
--> 1a7462d2b6d5
Step 7/7 : CMD ["/bin/bash"]
--> Using cache
--> fc966319ca5d
Successfully built fc966319ca5d
Successfully tagged f2f2508fbdcf9366289297fec219e6f8c9d5dd1:latest
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker inspect -f . f2f2508fbdcf9366289297fec219e6f8c9d5dd1

```

Figure 24: Jenkins docker logs.

### 3.4.2 Ansible

The main playbook, 'staged\_linux\_template.yaml,' as previously mentioned in section [3.1.3 Project Directory Structure](#), orchestrates the execution of all roles required by the pipeline for the creation and distribution of the templates. By

default, tasks within a playbook are executed in the order they appear, from top to bottom. However, the flow of execution can be altered using tags.

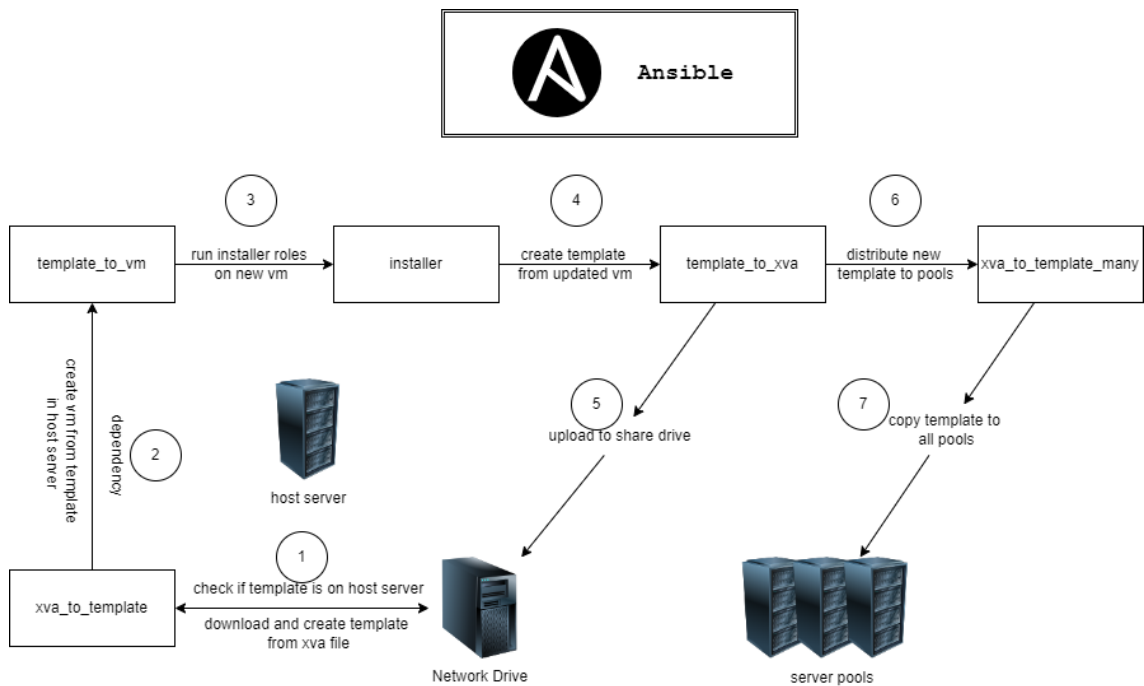


Figure 25: Ansible roles flow.

Figure 24 shows the flow of Ansible roles in the pipeline, the order of execution is marked with numbers. The purpose of this order is explained below in more detail, but it starts with the role that checks if base template is already existing on the host server where the virtual machine would be created. If not, it is downloaded from the network drive. Step 2 creates the virtual machine from this base template, after that installers are ran on the virtual machine leading to creating of updated template. After all this, the updated template is uploaded to the network drive and if the user opted for distribution the relevant role is also ran.

```

1  ---
2  - name: VM update and distribute
3    hosts: localhost
4
5    pre_tasks:
6      - name: setting facts
7        set_fact:
8          hostlist: "{{ hostlist_raw.split(',') | list }}"
9          tags: xva_to_template_many
10         when: "'xva_to_template_many' in ansible_run_tags"
11
12     roles:
13
14 >   - role: ../roles/template_to_vm...
15
16
17
18
19
20
21
22 >   - role: install_docker_nix...
23
24
25 >   - role: install_mssql_win...
26
27
28 >   - role: install_xcpngcenter_win...
29
30
31 >   - role: install_dotnet8...
32
33
34 >   - role: ../roles/template_to_xva...
35
36
37
38
39
40

```

Figure 26: Ansible playbook structure.

In this playbook, the pre-task 'setting facts' is executed only when the tag 'xva\_to\_template\_many' is specified during execution. This condition is enforced using the 'when' directive. Following the pre-tasks, the playbook lists all roles required for the deployment. Each role is assigned its own tag, with the tag name matching the role name for simplicity and clarity. This tagging system allows for fine-grained control over task execution.

The role **'template\_to\_vm'** contains the commands that are used on the host server which goes through several checks before creating a virtual machine. This role also has a dependency **'xva\_to\_template'** which does some prechecks and downloads the template from the shared network drive if it is not on the server already.

Once the virtual machine is successfully created, component installer roles such as `'install_mssql_win'` and `'install_dotnet8'` are executed based on the final template's requirements. Notably, the `'roles'` folder houses installer roles for both Windows and Linux systems, necessitating careful role selection.

Unneeded roles can be commented out or removed before repository push to GitLab.

Then after all the installations, the `'template_to_xva'` role starts the process of creating a template out of this updated virtual machine and then starts uploading it to the network share drive. Now that the updated template is in the network drive it can be accessed by all the host servers and can be copied easily.

### 3.5 Server Side

The server side consists of XenServer pools and the Virtual machines that are created in those pools. The XenServer cli commands enable Ansible modules to communicate with the servers and perform required actions.

```
---
- name: check uuid of local SR
  shell: xe sr-list content-type=user params=uuid | awk '{print $5}'
  register: sr_uuid
```

Figure 27: Ansible command to list uuid on XenServer using shell module.

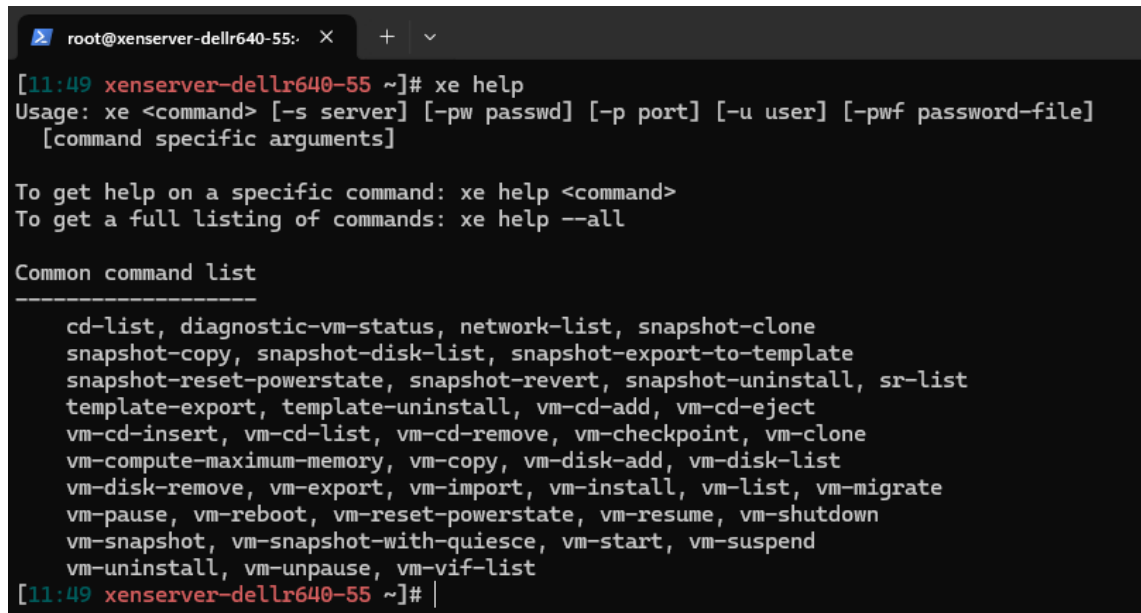
Using the shell module in Ansible is the same as running the commands directly on the target machine.

#### 3.5.1 XenServer CLI

Xe CLI is the command line interface for Citrix Hypervisor (XenServer) users. It allows for scripting and job automation management. The tool comes pre-installed by default on all Citrix Hypervisor servers and is part of XenCentre.

The CLI can be accessed by SSH-ing into any of the XenServers and provides a comprehensive and helpful interface for administrators to perform a wide

range of management tasks, including virtual machine management, resource allocation, network configuration, and system monitoring.



```

root@xenserver-dellr640-55: ~]# xe help
Usage: xe <command> [-s server] [-pw passwd] [-p port] [-u user] [-pwf password-file]
      [command specific arguments]

To get help on a specific command: xe help <command>
To get a full listing of commands: xe help --all

Common command list
-----
cd-list, diagnostic-vm-status, network-list, snapshot-clone
snapshot-copy, snapshot-disk-list, snapshot-export-to-template
snapshot-reset-powerstate, snapshot-revert, snapshot-uninstall, sr-list
template-export, template-uninstall, vm-cd-add, vm-cd-eject
vm-cd-insert, vm-cd-list, vm-cd-remove, vm-checkpoint, vm-clone
vm-compute-maximum-memory, vm-copy, vm-disk-add, vm-disk-list
vm-disk-remove, vm-export, vm-import, vm-install, vm-list, vm-migrate
vm-pause, vm-reboot, vm-reset-powerstate, vm-resume, vm-shutdown
vm-snapshot, vm-snapshot-with-quiet, vm-start, vm-suspend
vm-uninstall, vm-unpause, vm-vif-list
[11:49 xenserver-dellr640-55 ~]#

```

Figure 28: XE cli help section.

Some basic commands for virtual machine management involve:

- starting/stopping a vm
- cloning a vm
- creating a vm from a template
- creating a template from a vm

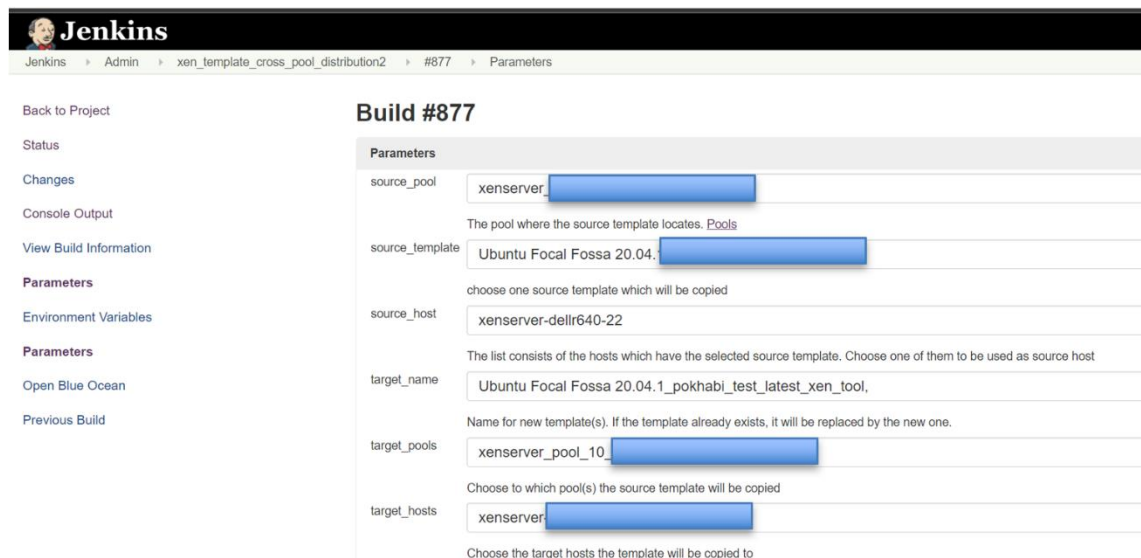


## 4 RESULTS AND DISUCSSION

This section highlights the results attained during the development of the Ansible and Jenkins solution. It also discusses the benefits of this solution compared to the old existing solution.

### 4.1 Old VS New

The old process of distributing templates also involves a Jenkins job but that job has limited functionality. It only copies the template from one pool to another. The normal workflow for creating a template requires a developer to find a template and create a virtual machine out of it via XCP-ng centre. Then manually update this template by installing all the required software components. Once done the developer would have to add the components to the description section in XCP-ng and then create a template out of it.



The screenshot shows the Jenkins interface for Build #877. The left sidebar contains navigation links: Back to Project, Status, Changes, Console Output, View Build Information, Parameters (highlighted), Environment Variables, Parameters, Open Blue Ocean, and Previous Build. The main content area is titled 'Build #877' and shows the 'Parameters' section. The parameters are as follows:

Parameter Name	Value	Description
source_pool	xenserver	The pool where the source template locates. Pools
source_template	Ubuntu Focal Fossa 20.04	choose one source template which will be copied
source_host	xenserver-dellr640-22	The list consists of the hosts which have the selected source template. Choose one of them to be used as source host
target_name	Ubuntu Focal Fossa 20.04.1_pokhabi_test_latest_xen_tool	Name for new template(s). If the template already exists, it will be replaced by the new one.
target_pools	xenserver_pool_10	Choose to which pool(s) the source template will be copied
target_hosts	xenserver	Choose the target hosts the template will be copied to

Figure 29: Old Jenkins template distribution job.

The old Jenkins pipeline is also only accessible by an internal IP address so developers would need to be connected to the internal VPN to use it.

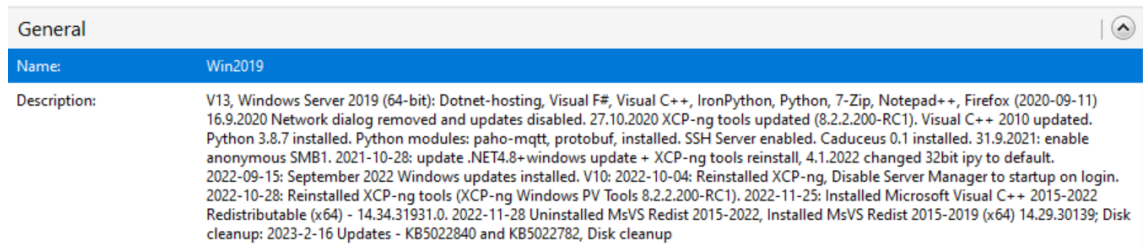


Figure 30: Old template description in XCP-ng.

Old Job	Ansible Solution
Developer must do software installation to the template on their own.	Ansible roles can be reused for templates.
Template can be distributed one pool at a time.	Multiple pools can be set for distribution.
Involves a lot of manual work.	Automated solution, reduces manual load.
Edit template description manually to remember what components have been installed	Uses git commit hash to track file in version control system which reflects which roles were installed.
Version tracking is difficult and based on names of the templates.	Template version will be reflected in git history.

The main advantage of using Ansible and role-based installation is that over long term, roles for crucial components for instance python can be devised in such a way that developer only needs to bump up the version number, example from 3.8 to 3.10 and rest of the role remains the same. This minor change if done manually over multiple templates and virtual machine ends up costing a lot of time of the developer and drains resources. Furthermore, such a solution is a step towards Infrastructure as Code type of development which is more suitable as a company scales up.

A rough estimate of hours spent on template updates is around 8-10hrs for creating new templates and then 35-40 for distribution to pools. Templates are updated at least quarterly so 4 times a year a developer spends around  $50 \times 4 = 200$  hours for this task. With Ansible the template creation and distribution time is combined and brought down to 1-2hrs for updating and 10-12 for distribution!

Thus, saving around  $200 - (14 \times 4) = 144$  hours. That is almost equivalent to one month's work!

## 4.2 Future plans and development

At the time of authoring this thesis the project is working as intended and has had multiple successful runs. Future plans for the project are to make other developers aware of its existence and start using it for the quarterly template updates happening every three months. It may also be used to develop and distribute templates needed by specific projects. Some planned changes that are already under development are:

- Template description should include link to GitLab commit so it is easier to track down updates.
- Implement some sort of version tracking in template naming i.e. Updated\_Template\_V1, Updated\_Template\_V2.
- Add date of creation to template description or template name.
- Create more Ansible roles for software components.

The most challenging task ahead would be to make other developers familiar with the Ansible playbooks and Jenkins pipelines since there are a lot of moving pieces in this project. Also, it is always difficult to get developers to adapt to something new especially in a bigger company where old practices are set in stone (almost).

## 5 CONCLUSION

This thesis documented the development of an automation solution for template updating and distribution within a company's virtual environment. It also covered DevOps tools like Ansible, Jenkins in depth and virtualization platform XCP-ng.

The main product of this thesis, a Jenkins pipeline was designed with the goal of reducing manual work while updating templates for development virtual machines and facilitate their deployment in all server pools. Ansible was used to automate the process and create roles and playbooks which could be reused and save up on roughly 140+ hours. Using Ansible was also in line with Infrastructure as Code development strategy which emphasizes on having definitions in version control system rather than maintaining the code through manual process. The Jenkins pipeline created could be accessed easily and did not require a VPN connection like the older solution and was clean, intuitive, and user-friendly.

Although the pipeline has not been widely taken in use yet, there are future plans for that, and the initial response of the developers has been positive with some room for development. Suggestions include features such as more informative template description, automated versioning, and creation of more installer components as Ansible roles.

## REFERENCES

Ansible. (n.d.). Official Ansible Documentation. Read on 04.12.2023 . Retrieved from <https://docs.ansible.com/>

Ansible. (n.d.). How Ansible Works. Read on 08.12.2023. Retrieved from <https://www.ansible.com/overview/how-ansible-works>

Ansible. (n.d.). Ansible Playbook User Guide. Read on 08.12.2023. Retrieved from [https://docs.ansible.com/ansible/latest/playbook\\_guide/index.html](https://docs.ansible.com/ansible/latest/playbook_guide/index.html)

Ansible. (n.d.). Ansible Roles User Guide. Read on 10.01.2024. Retrieved from [https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_reuse\\_roles.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html)

DevOpsSchool. (n.d.). Understanding Ansible Architecture Using Diagram. Read on 10.01.2024. Retrieved from <https://www.devopsschool.com/blog/understanding-ansible-architecture-using-diagram/>

Jenkins (software). (n.d.). In Wikipedia. Read on 12.12.2024. Retrieved from [https://en.wikipedia.org/wiki/Jenkins\\_\(software\)](https://en.wikipedia.org/wiki/Jenkins_(software))

Jenkins Plugins. (n.d.). Read on 15.02.2024. Retrieved from <https://plugins.jenkins.io/>

Jenkins. (n.d.). Managing Nodes Components of Distributed Builds: Creating Agents. Read on 03.03.2024. Retrieved from <https://www.jenkins.io/doc/book/managing/nodes/#creating-agents>

LinuxConfig.org. (n.d.). XE Full Command List Reference with Description for XenServer. Read on 17.11.2023. Retrieved from <https://linuxconfig.org/xs-full-command-list-reference-with-description-for-xenserver>

XCP-ng. (n.d.). XCP-ng Management: Manage Locally: CLI. Retrieved from <https://docs.xcp-ng.org/management/manage-locally/cli/>