Contents

[Ansible 5](#_Toc103071241)

[Why Ansible? 5](#_Toc103071242)

[What is Ansible? 7](#_Toc103071243)

[Pull Configuration 8](#_Toc103071244)

[Push Configuration 8](#_Toc103071245)

[Ansible Architecture 9](#_Toc103071246)

[Playbooks 10](#_Toc103071247)

[Inventory Management 14](#_Toc103071248)

[How Ansible Works in the Real World 14](#_Toc103071249)

[Alternatives to Ansible 16](#_Toc103071250)

[History of Ansible 16](#_Toc103071251)

[Important Terms Used in Ansible 17](#_Toc103071252)

[What is Configuration Management 17](#_Toc103071253)

[How Ansible Works? 17](#_Toc103071254)

[Installing Ansible 19](#_Toc103071255)

[Installation Process 19](#_Toc103071256)

[Control Machine Requirements 19](#_Toc103071257)

[Install Ansible on Linux - Centos/RedHat systems 19](#_Toc103071258)

[Install Ansible on Ubuntu/Debian systems 19](#_Toc103071259)

[Install Ansible on Windows 20](#_Toc103071260)

[Enabling Ubuntu on Windows 10 – WSL (*Windows Subsystem for Linux*) 20](#_Toc103071261)

[Run Ansible on Windows 10: Using Cygwin 21](#_Toc103071262)

[Run Ansible on Windows 10: Using a Linux VM 26](#_Toc103071263)

[YAML Basics 37](#_Toc103071264)

[Understanding YAML 37](#_Toc103071265)

[key-value pair 37](#_Toc103071266)

[Representing List 37](#_Toc103071267)

[List inside Dictionaries 38](#_Toc103071268)

[List of Dictionaries 38](#_Toc103071269)

[Some common words related to Ansible. 39](#_Toc103071270)

[Ad-hoc Commands 40](#_Toc103071271)

[Parallelism and Shell Commands 40](#_Toc103071272)

[File Transfer 40](#_Toc103071273)

[Transferring file to many servers/machines 40](#_Toc103071274)

[Creating new directory 40](#_Toc103071275)

[Deleting whole directory and files 40](#_Toc103071276)

[Managing Packages 40](#_Toc103071277)

[Gathering Facts 41](#_Toc103071278)

[More Ad-hoc Commands 41](#_Toc103071279)

[Hosts Accessible 41](#_Toc103071280)

[Copy File 42](#_Toc103071281)

[Install a package via the yum module on two Centos hosts 43](#_Toc103071282)

[Remove the previously installed ncdu package 45](#_Toc103071283)

[Gather some facts about the system 47](#_Toc103071284)

[Dry-run p4.yml playbook 47](#_Toc103071285)

[Run p4.yml playbook with password authentication for all hosts 47](#_Toc103071286)

[Ansible Playbooks 48](#_Toc103071287)

[Playbook Structure 48](#_Toc103071288)

[Create a Playbook 48](#_Toc103071289)

[The Different YAML Tags 49](#_Toc103071290)

[name 49](#_Toc103071291)

[hosts 49](#_Toc103071292)

[vars 49](#_Toc103071293)

[tasks 49](#_Toc103071294)

[Examples 49](#_Toc103071295)

[One playbook with one play 49](#_Toc103071296)

[Two plays for two hosts 51](#_Toc103071297)

[Task Calls Upon Another Task 52](#_Toc103071298)

[Ansible Roles 54](#_Toc103071299)

[Creating a New Role 55](#_Toc103071300)

[Role Structure 55](#_Toc103071301)

[Usage 55](#_Toc103071302)

[Options 55](#_Toc103071303)

[Creating a Role Directory 55](#_Toc103071304)

[Utilizing Roles in Playbook 56](#_Toc103071305)

[Breaking a Playbook into a Role 57](#_Toc103071306)

[Output 57](#_Toc103071307)

[Ansible Variables 67](#_Toc103071308)

[Usage of variable - {{Output}} 67](#_Toc103071309)

[Exception Handling in Playbooks 68](#_Toc103071310)

[Loops 68](#_Toc103071311)

[Blocks 69](#_Toc103071312)

[Conditionals 69](#_Toc103071313)

[Ansible - Advanced Execution 70](#_Toc103071314)

[How to Limit Execution by Tasks 70](#_Toc103071315)

[How to Limit Execution by Hosts 70](#_Toc103071316)

[Alternate Solution 71](#_Toc103071317)

[Running the Playbook 71](#_Toc103071318)

[Playbook targeting a single host 71](#_Toc103071319)

[Ansible Troubleshooting 72](#_Toc103071320)

[Debug and Register 72](#_Toc103071321)

[Use Verbosity 72](#_Toc103071322)

[Important Points 72](#_Toc103071323)

[Solution 72](#_Toc103071324)

[Common Playbook Issues 73](#_Toc103071325)

[Hands-on Lab 75](#_Toc103071326)

[1. Create an inventory 76](#_Toc103071327)

[2. Creating a Directory Structure and your Playbook 76](#_Toc103071328)

[3. Run it 77](#_Toc103071329)

[4. Extend the Playbook 78](#_Toc103071330)

[5. Extend the playbook more: copy files 78](#_Toc103071331)

[6. From 1 to Many 79](#_Toc103071332)

[7. Adding Variables to the Mix 79](#_Toc103071333)

[8. Decide what to do - using conditionals 81](#_Toc103071334)

[9. Using Loops 82](#_Toc103071335)

[10. Templates 82](#_Toc103071336)

[References 84](#_Toc103071337)

# Ansible

**Ansible** is an open-source automation and orchestration tool for software provisioning, configuration management, and software deployment. Ansible can easily run and configure Unix-like systems as well as Windows systems to provide infrastructure as code. It contains its own declarative programming language for system configuration and management.

Ansible is an Infrastructure as Code tool used for managing and monitoring remote servers. It is one of the key tools you would have in your DevOps environment.

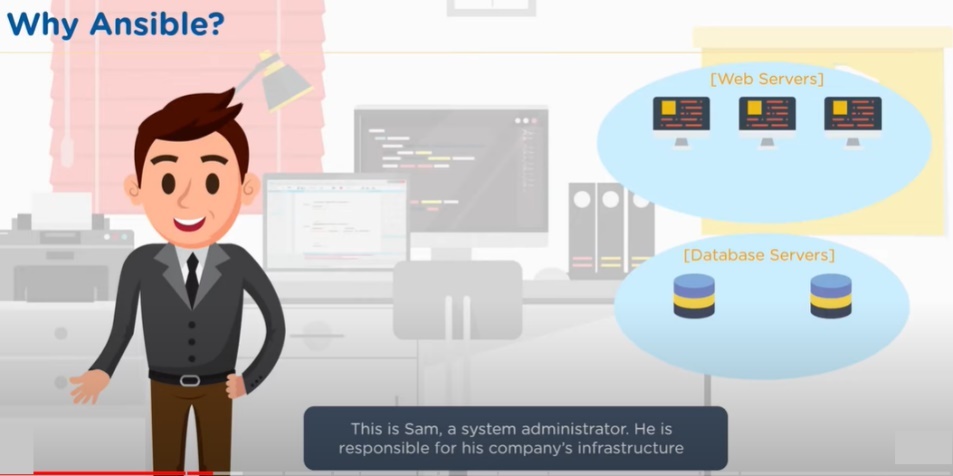
Ansible is popular for its simplicity of installation, ease of use in what concerns the connectivity to clients, its lack of agent for Ansible clients and the multitude of skills. It functions by connecting via SSH to the clients, so it doesn’t need a special agent on the client-side, and by pushing modules to the clients, the modules are then executed locally on the client-side and the output is pushed back to the Ansible server.

Since it uses SSH, it can very easily connect to clients using SSH-Keys, simplifying though the whole process. Client details, like hostnames or IP addresses and SSH ports, are stored in files called inventory files. Once you have created an inventory file and populated it, ansible can use it.

## Why Ansible?

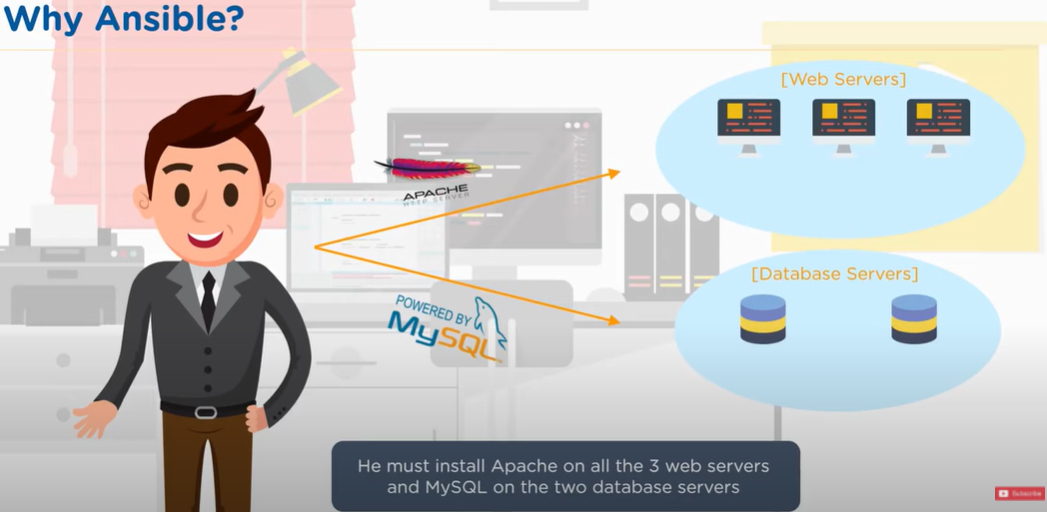
Here are some important pros/benefits of using Ansible:

* One of the most significant advantages of Ansible is that it is free to use by everyone.
* It does not need any special system administrator skills to install and use Ansible, and the official documentation is very comprehensive.
* Its modularity regarding plugins, modules, inventories, and playbooks make Ansible the perfect companion to orchestrate large environments.
* Ansible is very lightweight and consistent, and no constraints regarding the operating system or underlying hardware are present.
* It is also very secure due to its agentless capabilities and due to the use of OpenSSH security features.
* Another advantage that encourages the adoption of Ansible is its smooth learning curve determined by the comprehensive documentation and easy to learn structure and configuration.



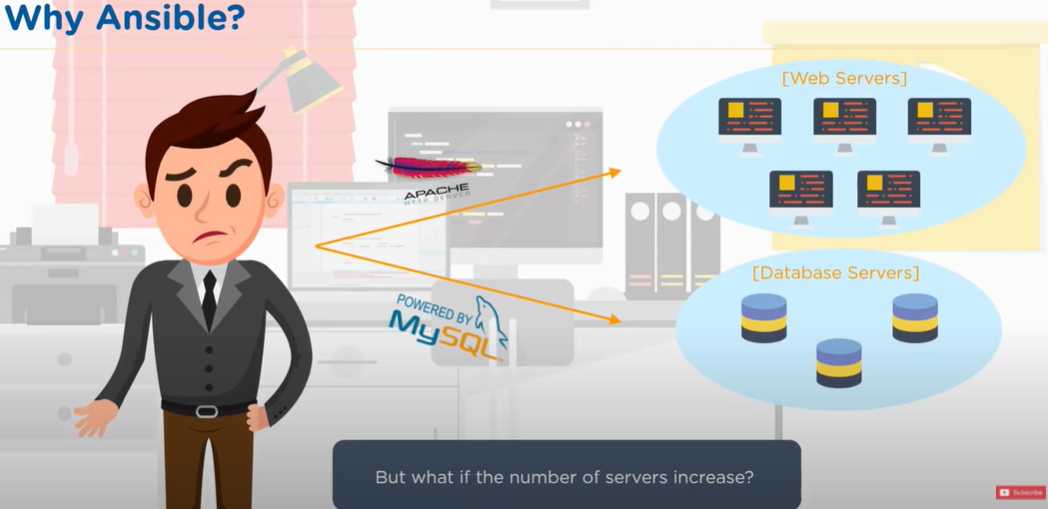
This is on the operations side of the DevOps equation.

Some of the services Sam must maintain could be webservers running Apache, database servers etc.



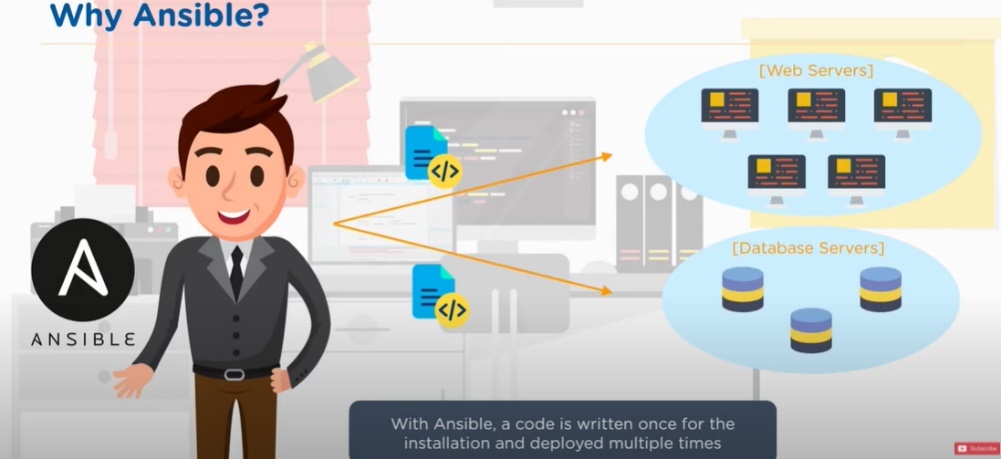
Having a few servers is easy to maintain. Wouldn’t take much time either.

But what if the number of servers increases, which is a reality? Then it gets tricky.



The same task must be repeated multiple times. Moreover, humans are prone to make errors. If done manually, there is a good chance that all servers are not setup identically.

This is where Ansible comes to the rescue and helps you become an efficient operations team!

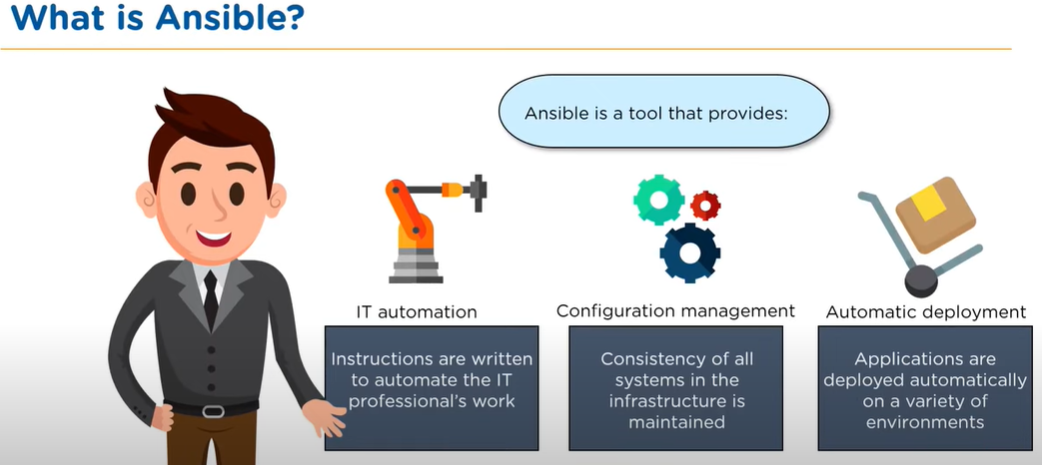


With Ansible, code is written once for the installation and deployed multiple times. Write one script and then each script can be executed and have a consistent environment.

So now, Sam, the DevOps person, can work on more productive tasks rather than the repetitive ones.

## What is Ansible?

Ansible allows you to create and control three key areas that you would have within your operations environment.



First, there is IT Automation. So, you can actually write instructions that automate the IT setup that you would typically do manually in the past.

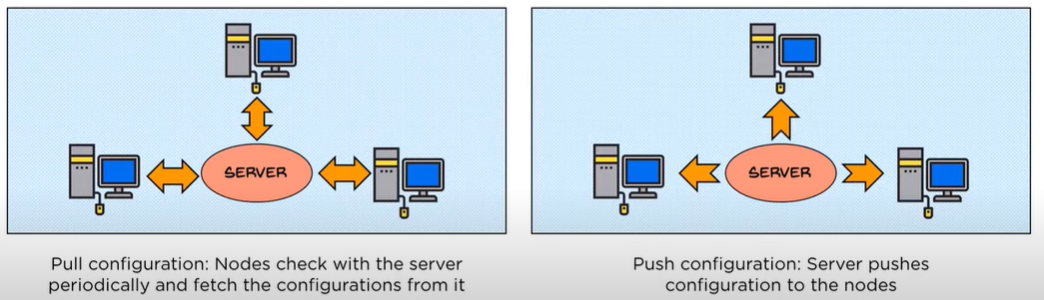
The second is the Configuration and having consistent configuration. Imagine setting up hundreds of Apache servers and being able to guarantee with precision that each of those Apache servers is setup identically.

And then finally, you want to be able to automate the deployment so that as you scale up your server environment, you can just push out instructions that can deploy automatically different servers.

The bottom line is, you want to be able to speed up and make your operations team more efficient.

### Pull Configuration

There are 2 different ways to setup environments for server farms.



One is to have a key server that has all the instructions on and then on each of the servers that connect to that main master server, you would have a piece of software known as a client, installed on each of those servers that would communicate to the main master server and then we periodically either update or change the config of the slave/worker server.

This is known as a “Pull Configuration”.

### Push Configuration

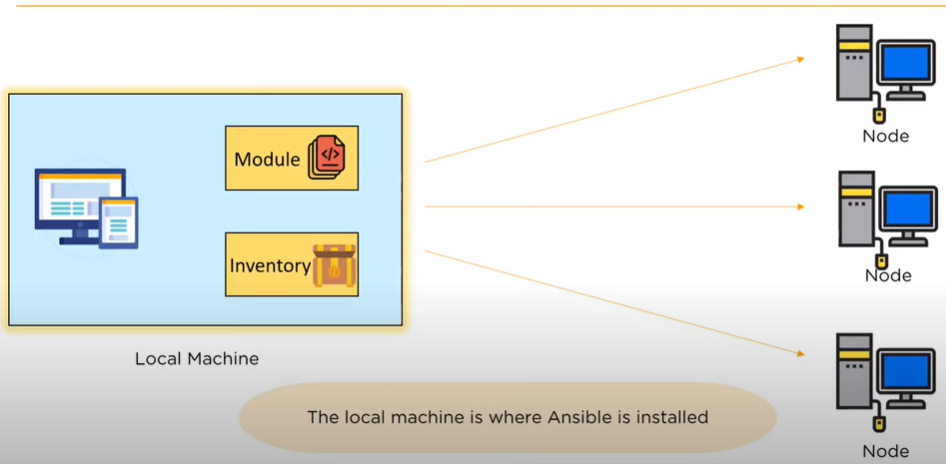
An alternative is a push configuration. And the push config is slightly different. The main difference is as with a pull config, you have a master server where you actually put up the instructions, but unlike the pull config, where you have a client installed on each of the services, with a push config, you actually have no client installed on the remote servers.

You simply are pushing out the config to those servers and forcing a restructure or a fresh clean installation in that environment.

So, Ansible is one of those second environs where it is a push config server and this contrasts other popular products like Chef and Puppet, which have a master-slave architecture with a master server connecting with a client on a remote slave environment where you would then be pushing out the update, as well you’re pushing out the service and the structure of the server to remote hardware and you are just putting it onto the hardware, irrelevant of the structure that’s out there.

And there are some significant advantages that you have in that you’re not having to have the extra overhead weight of a client installed on those remote servers, having to constantly communicate back to the master environment.

## Ansible Architecture

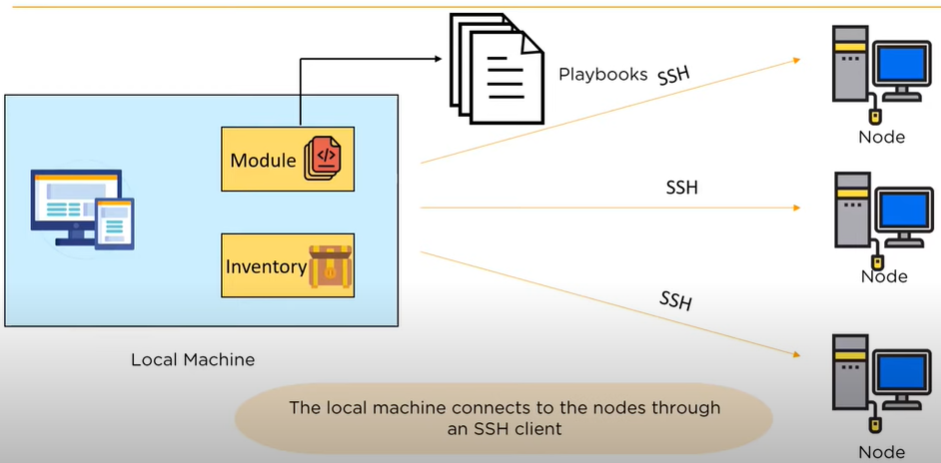


The **Local Machine** is where you will have all your instructions and really the power of the control that you would be pushing out to the remote servers. So the local machine is where you will be starting and doing all of your work.

Connected from the local machine, are all the different **nodes** pushing out the difference configs that you would setup on the local machine.

The configs that you would write, you would write those in code within a module. So, you do this on your local machine for creating these modules and each of these modules is actually consistent **playbooks**

The local machine also has a second job, and that job is to manage the inventory of the nodes that you have in your environment.



The local machine is able to connect to each of the different nodes that you would have in your hardware network through **SSH clients** or a secure client.

Let’s take a look at some of the different elements within the architecture. First, let’s take a look at **Playbooks** that you would write and create for the Ansible environments.

### Playbooks

The core of Ansible is the Playbook.

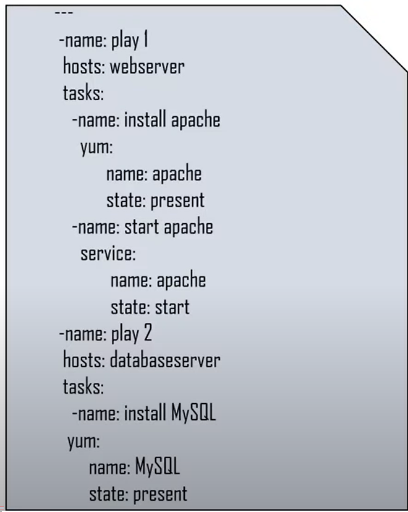
Graphical user interface

Description automatically generated

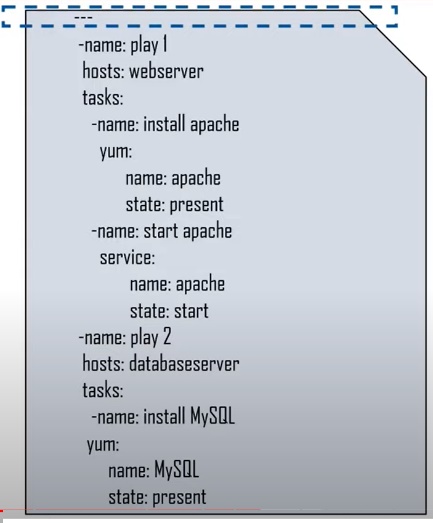
This is where you create the instructions that you write to define the architecture of your hardware. So, the Playbook is really just a set of instructions that configure the different nodes that you have and each of those set of instructions in a language called **YAML**. This is a standard language used for config server environments.

YAML (/ˈjæməl/, rhymes with *camel*) was first proposed by Clark Evans in 2001, who designed it together with Ingy döt Netand Oren Ben-Kiki. Originally YAML was said to mean *Yet Another Markup Language*, because it was released in an era that saw a proliferation of markup languages for presentation and connectivity (HTML, XML, SGML, etc). Its initial name was intended as a tongue-in-cheek reference to the technology landscape, referencing its purpose as a markup language with the yet another construct, but it was then repurposed as *YAML Ain't Markup Language*, a recursive acronym, to distinguish its purpose as data-oriented, rather than document markup.

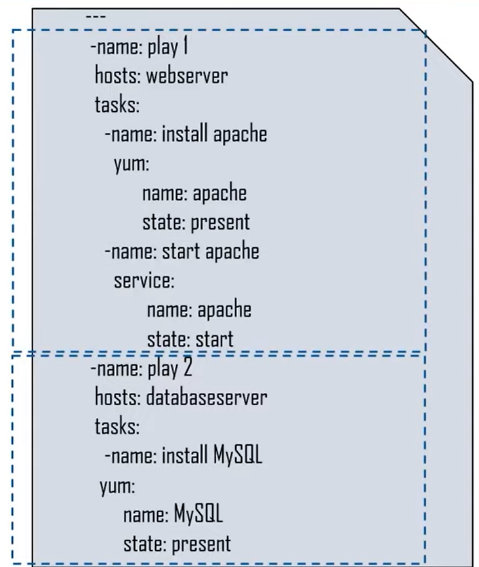
* Here is a sample playbook structure:



* You start off your YAML script with **3 dashes (---)** and that indicates the **start** of a script.



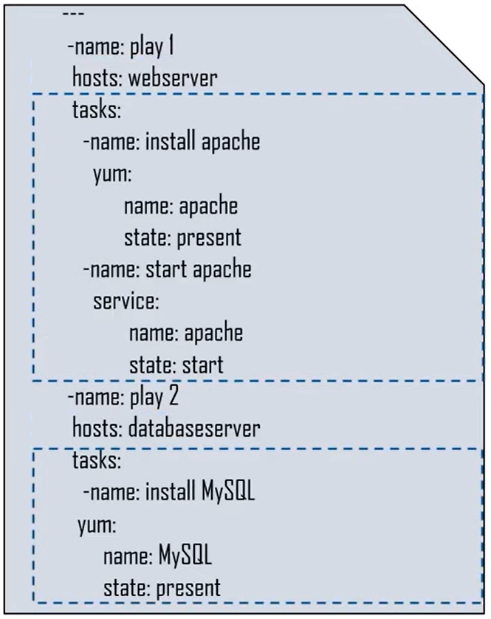
* Then the script itself is actually consistent of two distinct **plays**. At the top, we have “Play 1” and below that, we have “Play 2”.



* Within each of those plays, we define which **nodes** are we targeting. So, here we have a web server in the top play and in the second play, we have a database server that we are targeting.

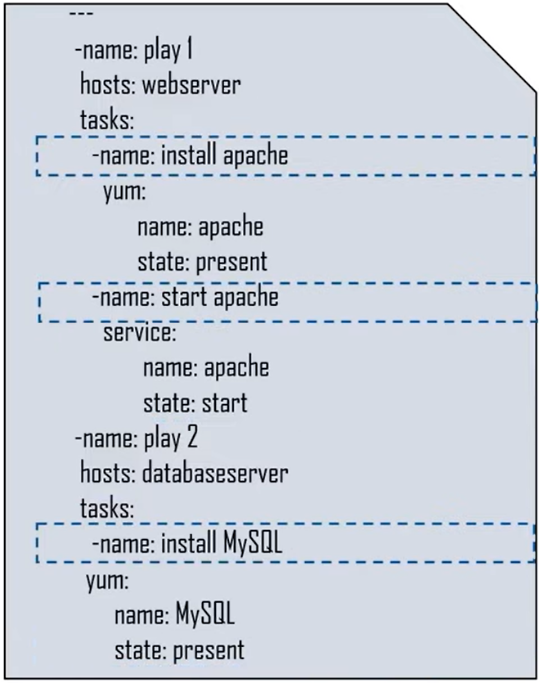


* And then within each of those server environments, we have the specific **tasks** that we are looking to execute.



So, let’s step through some of these tasks.

* We have an “install apache” task, a “start apache” task and have an “install MySQL” task.



* When we do that, we are going to execute specific set of instructions and those instructions can include installing Apache, and then setting the state of the Apache environments or starting the Apache environment and setting up and running the MySQL environment.

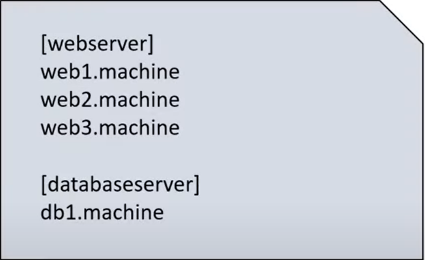


So, this really isn’t too complicated and that’s the really good thing about YAML is it is really designed to make it easy for you as an Operations Lead to be able to configure the environments that you want to consistently create.

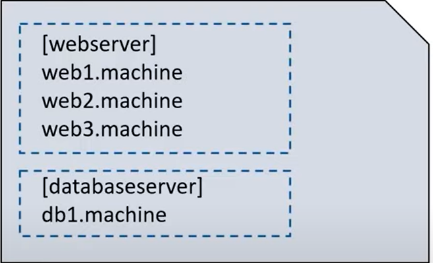
Let’s take a step back. We have 2 hosts, a “webserver” and a “databaseserver”. Where do these names come from? This takes us into our next stage and the second part of working with Ansible, which is the Inventory Management part of Ansible.

### Inventory Management

The inventory part of Ansible is where we maintain the structure of our network environment. So, what we do here is part of the structure in creating different nodes is we’ve had to create two different nodes here. We have a “webserver” node and a “databaseserver” node.



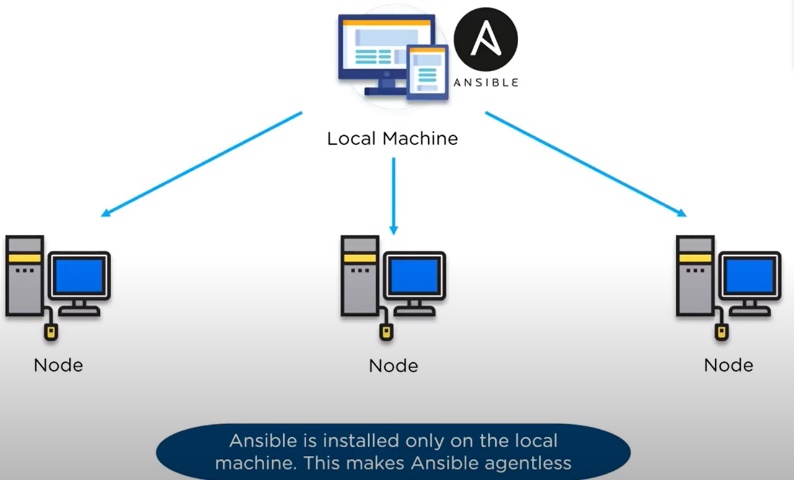
And under webserver node, we have the names that we are pointing to specific machines within that environment. So, when we write our script, all we have to do is refer to either webserver or database server, and the different servers will have the instructions from the YAML script executed on them.



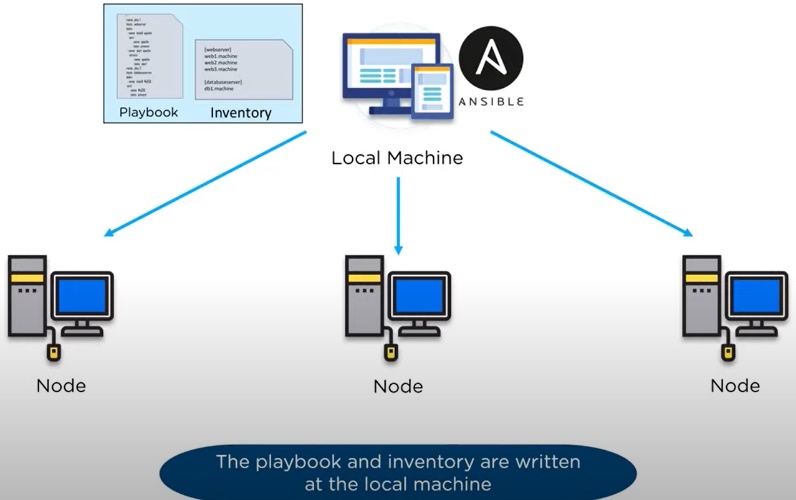
This makes it really easy for you to be able to just point to new services without having to write out complex instructions.

## How Ansible Works in the Real World

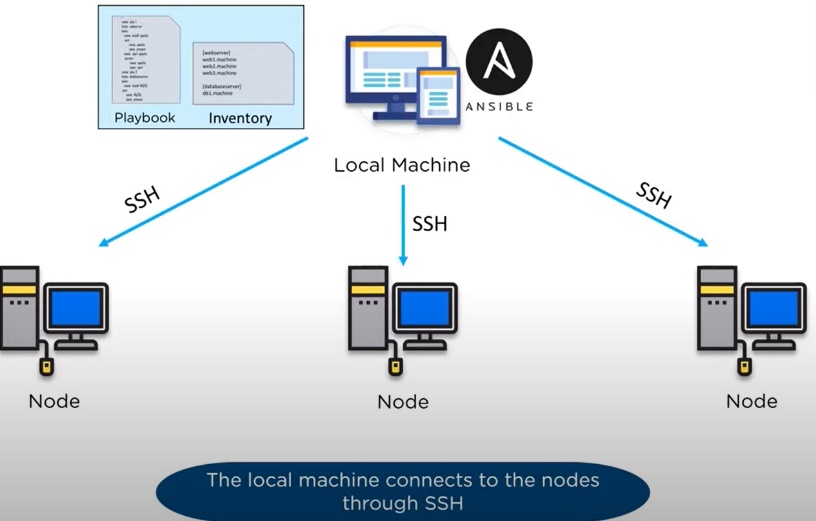
* You will have the Ansible software installed on the local machine. It connects to the different nodes within your network.



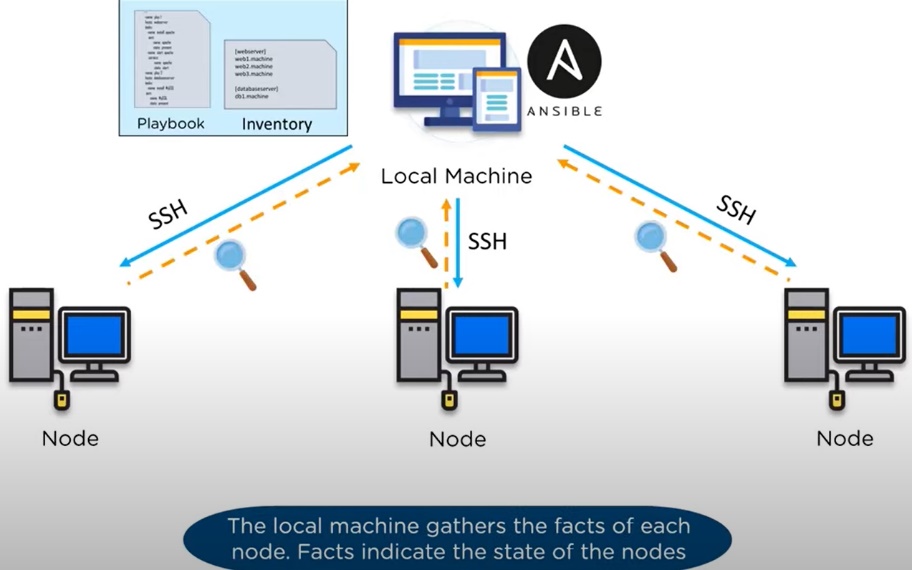
* On the local machine, you will have your playbook, which is the set of instructions for how to setup the remote nodes and then to identify how you’re going to connect those nodes, you will have an inventory.

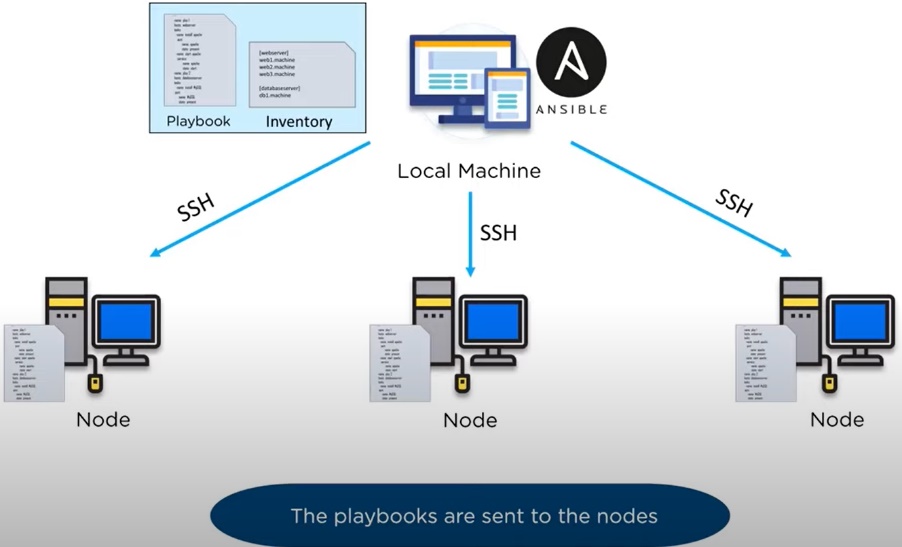


* We use secure SSH connections to each of the servers, so we are encrypting the communication to those servers

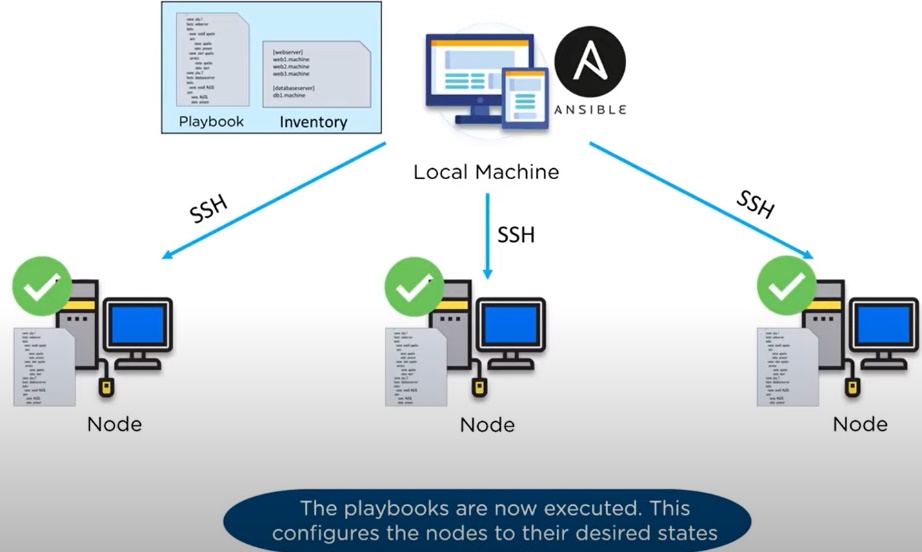


* We are able to grab some basic facts on each server so we understand how we can then push out the playbook each server and configure that server remotely.





* The end goal is to have an environment that is consistent.



## Alternatives to Ansible

While there are many alternative to Ansible (Chef, Puppet) out there that do the same thing with some differences, Ansible managed to rise above all of them with its simplicity, improved security, and most important its smooth learning curve. Due to these qualities and fast adoption of Ansible, we created a tutorial full of examples so you can have an even more seamless first experience in working with Ansible.

## History of Ansible

Here, are important landmarks from the history of ansible:

* In February 2012 the Ansible project began. It was first developed by Michael DeHaan, the creator of Cobbler and Func (Fedora Unified Network Controller).
* Initially called AnsibleWorks Inc, the company funding the ansible tool was acquired in 2015 by RedHat and later on, along with RedHat, moved under the umbrella of IBM.
* In the present, Ansible comes included in distributions like Fedora Linux, RHEL, Centos and Oracle Linux.

## Important Terms Used in Ansible

* **Ansible server**: The machine where Ansible is installed and from which all tasks and playbooks will be ran.
* **Module**: Basically, a module is a command or set of similar Ansible commands meant to be executed on the client-side.
* **Task**: A task is a section that consists of a single procedure to be completed.
* **Role**: A way of organizing tasks and related files to be later called in a playbook.
* **Fact**: Information fetched from the client system from the global variables with the gather-facts operation.
* **Inventory**: File containing data about the ansible client servers. Defined in later examples as hosts file.
* **Play**: Execution of a playbook.
* **Handler**: Task which is called only if a notifier is present.
* **Notifier**: Section attributed to a task which calls a handler if the output is changed.
* **Tag**: Name set to a task which can be used later on to issue just that specific task or group of tasks.

## What is Configuration Management

Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise’s hardware and software.

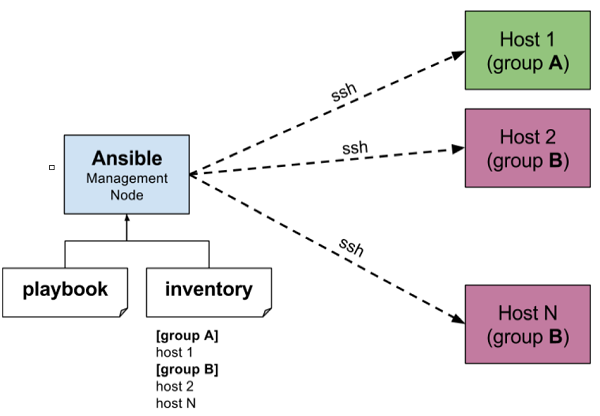
Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of **WebLogic/WebSphere** server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

## How Ansible Works?

The picture given below shows the working of Ansible.

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



The management node in the above picture is the controlling node (managing node) which controls the entire execution of the playbook. It’s the node from which you are running the installation. The inventory file provides the list of hosts where the Ansible modules needs to be run and the management node does a SSH connection and executes the small modules on the hosts machine and installs the product/software.

**Beauty** of Ansible is that it removes the modules once those are installed so effectively it connects to host machine , executes the instructions and if it’s successfully installed removes the code which was copied on the host machine which was executed.



# Installing Ansible

## Installation Process

Mainly, there are two types of machines when we talk about deployment −

* **Control machine** − Machine from where we can manage other machines.
* **Remote machine** − Machines which are handled/controlled by control machine.

There can be multiple remote machines which are handled by one control machine. So, for managing remote machines we have to install Ansible on control machine.

## Control Machine Requirements

Ansible can be run from any machine with Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed.

**Note** − Windows does not support control machine.

By default, Ansible uses **ssh** to manage remote machine.

Ansible does not add any database. It does not require any daemons to start or keep it running. While managing remote machines, Ansible **does not** leave any software installed or running on them. Hence, there is no question of how to upgrade it when moving to a new version.

Ansible can be installed on control machine which have above mentioned requirements in different ways. You can install the latest release through Apt, yum, pkg, pip, OpenCSW, pacman, etc.

## Install Ansible on Linux - Centos/RedHat systems

**Step 1)** Install EPEL repo

[root@ansible-server ~]# sudo yum install epel-release

**Step 2) Install ansible package**

[root@ansible-server ~]# sudo yum install -y ansible

Text

Description automatically generated

## Install Ansible on Ubuntu/Debian systems

**Step 1)** Perform an update to the packages

$ sudo apt update

**Step 2)** Install the software-properties-common package

$ sudo apt install software-properties-common

**Step 3)** Install ansible personal package archive

$ sudo apt-add-repository ppa:ansible/ansible

**Step 4)** Install ansible

$ sudo apt update

$ sudo apt install ansible

## Install Ansible on Windows

<https://phoenixnap.com/kb/install-ansible-on-windows#htoc-method-3-enabling-ubuntu-on-windows>

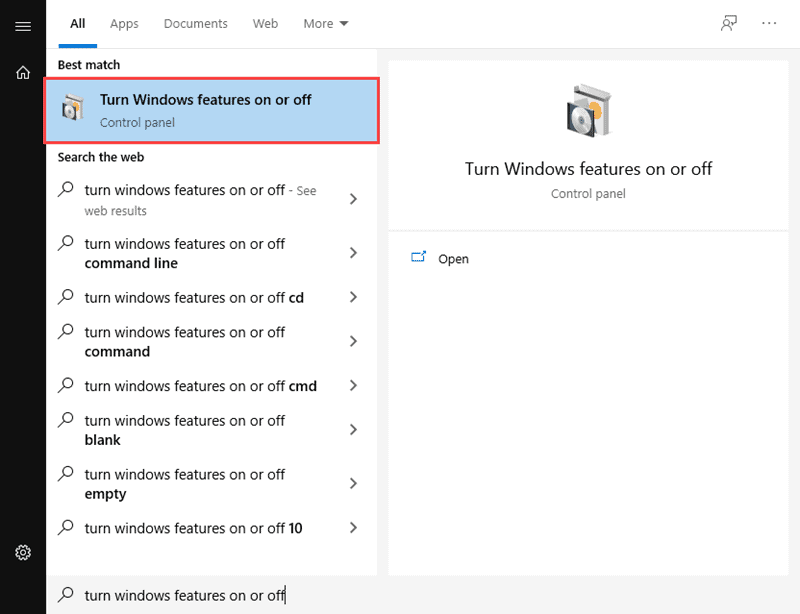
Ansible is an Infrastructure as Code tool used for managing and monitoring remote servers.

Ansible requires a Linux-based system to run. This can be a problem if you want to use its features on a Windows 10 system. However, there are ways to install Ansible on Windows.

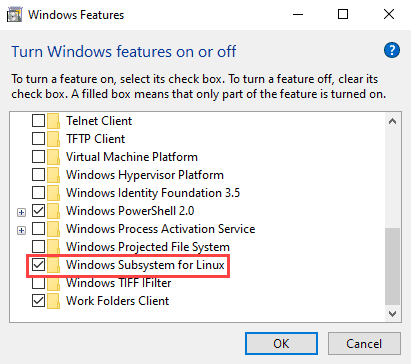
### Enabling Ubuntu on Windows 10 – WSL (*Windows Subsystem for Linux*)

Use the Windows Subsystem for Linux to start up the Ubuntu terminal without setting up a virtual machine:

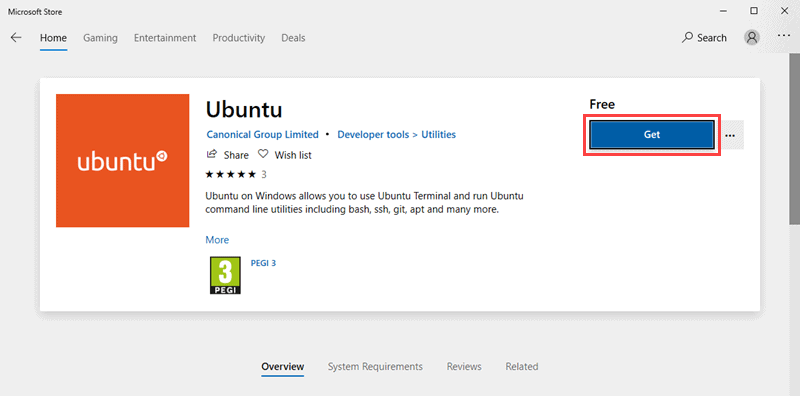
1. Open the **Start** menu and search for Turn Windows features on or off. Click on the shortcut when it appears.



1. Scroll down through the list of features until you see Windows Subsystem for Linux. Click on the checkbox, and then click **OK** to enable the feature.



1. Open the Microsoft Store. Search for Ubuntu and click on **Get** to install the latest version.



1. 4. Once the installation is complete, click on **Launch** to start up the Ubuntu command terminal.
2. 5. To install Ansible, enter the following set of commands:

$ sudo apt-get update

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository ppa:ansible/ansible

$ sudo apt-get update

$ sudo apt-get install ansible -y

After running the above line of code, you are ready to manage remote machines through Ansible. Just run ansible –version to check the version and just to check whether Ansible was installed properly or not.

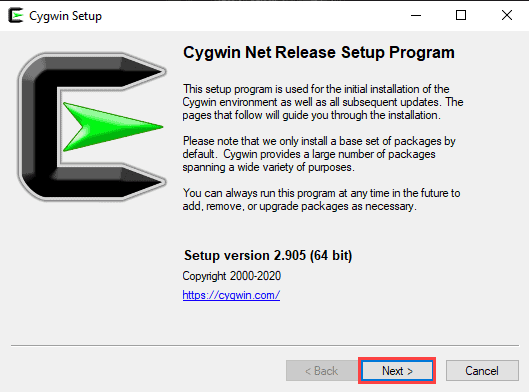
### Run Ansible on Windows 10: Using Cygwin

Cygwin is a POSIX-compatible environment that lets you run tools and code designed for Unix-like operating systems on Microsoft Windows.

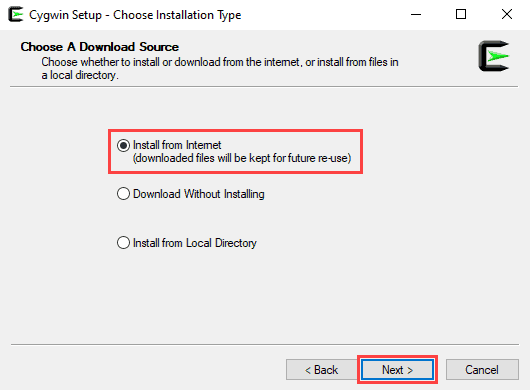
Even though the default Cygwin installation contains hundreds of tools for Unix-based systems, Ansible is not one of them. You must manually add Ansible during the installation process.

To install Ansible on Windows using Cygwin, follow these steps:

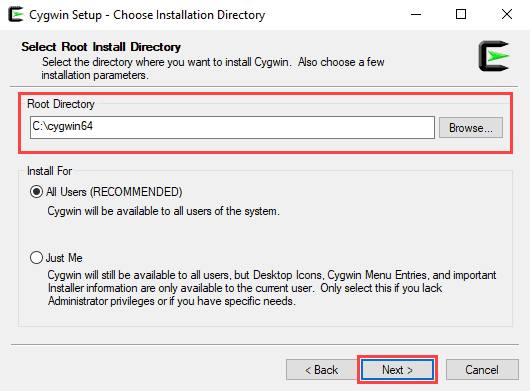
1. Download the [Cygwin installation file](https://cygwin.com/setup-x86_64.exe) (<https://cygwin.com/setup-x86_64.exe>) This file is compatible with both the 32-bit and 64-bit versions of Windows 10. It automatically installs the right version for your system.
2. Run the Cygwin installation file. On the starting screen of the installation wizard, click **Next** to continue.



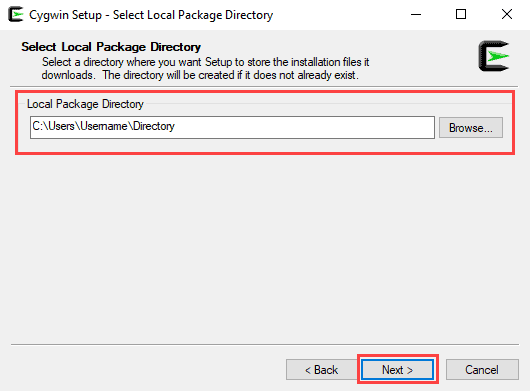
1. Select **Install from Internet** as the download source and click **Next**.



1. In the *Root Directory* field, specify where you want the application installed, then click **Next**.

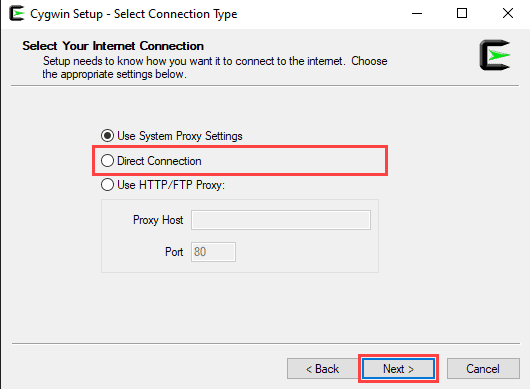


1. In the *Local Package Directory* field, select where you want to install your Cygwin packages, then click **Next**.

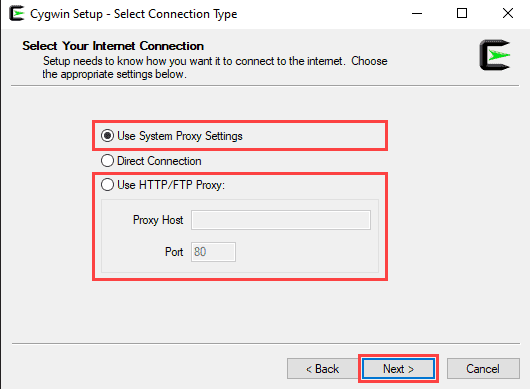


1. Choose the appropriate Internet connection option.

If you aren’t using a proxy, select **Direct Connection**.

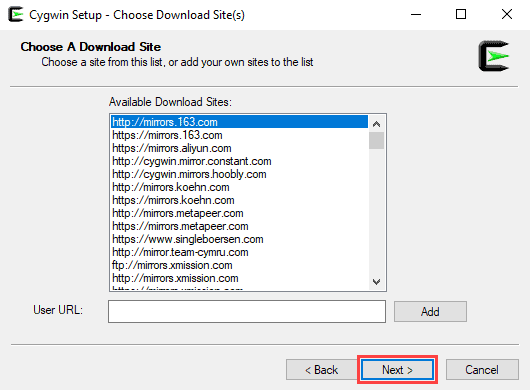


If you are using a proxy, select **Use System Proxy Settings** or enter the proxy settings manually with the **Use HTTP/FTP Proxy.**



Click **Next** to continue.

1. Choose one of the available mirrors to download the installation files, then click **Next**.

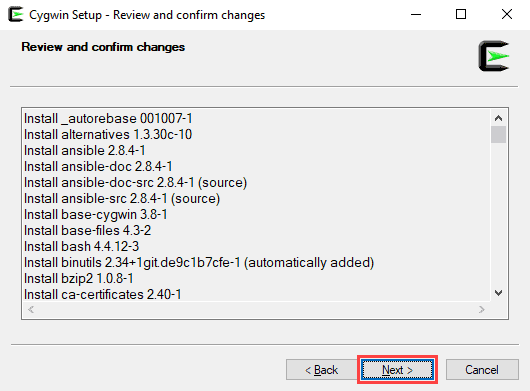


1. On the *Select Packages* screen, change the **View** option to **Full** and type ‘ansible’ in the search bar.

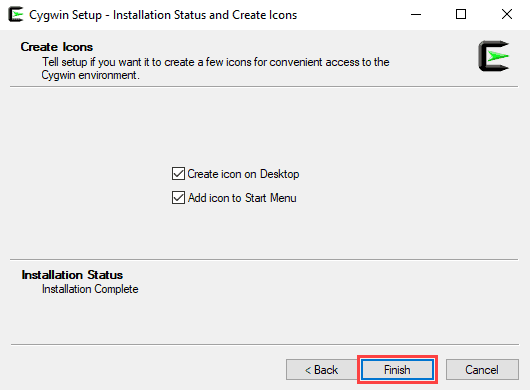
Select both **Ansible** and **Ansible Doc** by checking the boxes under *Src?* and click **Next** to continue.



1. This screen lets you review the installation settings. To confirm and begin the install process, click on **Next**.



1. The install wizard will download and install all the selected packages, including Ansible.
2. Once the installation is complete, select whether you want to add a Cygwin desktop and Start Menu icon, then click on **Finish** to close the wizard.



### Run Ansible on Windows 10: Using a Linux VM

Another way to install Ansible on Windows 10 is to use a virtualization tool and a Linux virtual box.

In this example, we will use Oracle VM VirtualBox to set up an Ubuntu virtual machine and install Ansible.

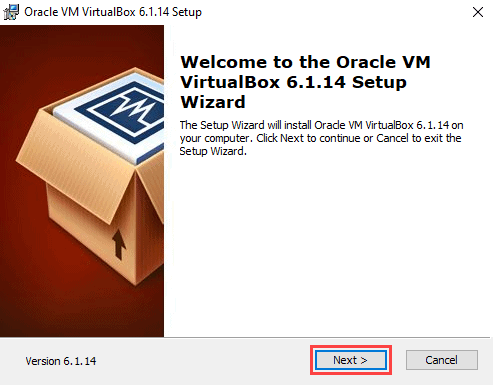
#### Step 1: Setting Up VirtualBox

1. Download the [VirtualBox installation file](https://download.virtualbox.org/virtualbox/6.1.14/VirtualBox-6.1.14-140239-Win.exe):

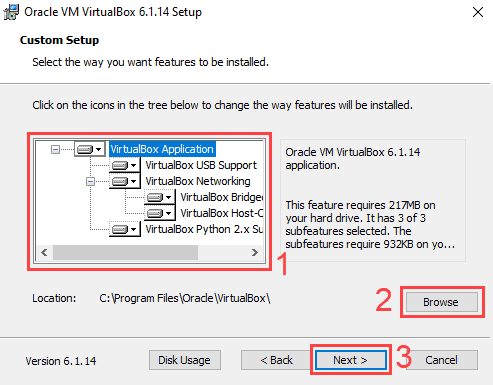
(<https://download.virtualbox.org/virtualbox/6.1.14/VirtualBox-6.1.14-140239-Win.exe>)

1. Run the VirtualBox installation file.

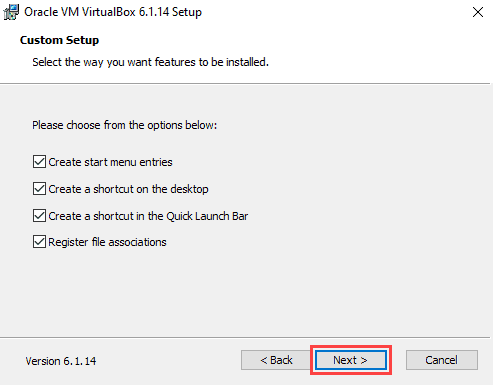
On the starting screen of the installation wizard, click **Next** to continue.



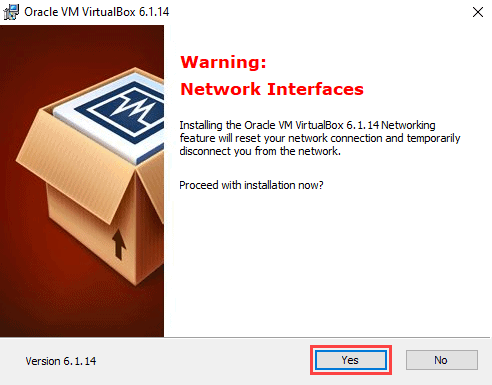
1. Use this screen to select which features of VirtualBox you want to install. This is also where you can set the install location. Click **Next** to continue.



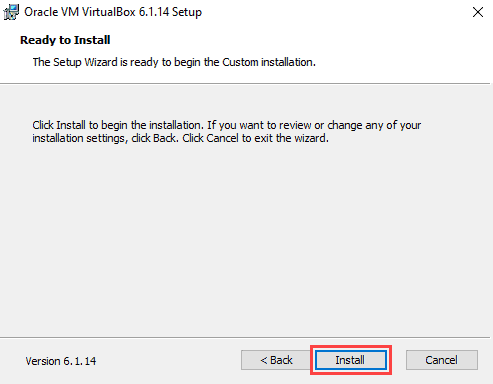
1. Choose whether you want to add Desktop, Start Menu, and Quick Launch Bar shortcuts and add file associations. Click **Next** to continue.



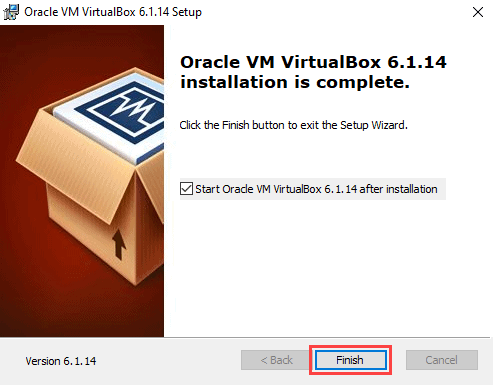
1. A new screen will warn you that the install wizard needs to reset your network connection. Click **Yes** to continue.



1. On the next screen, click **Install** to begin the installation process.

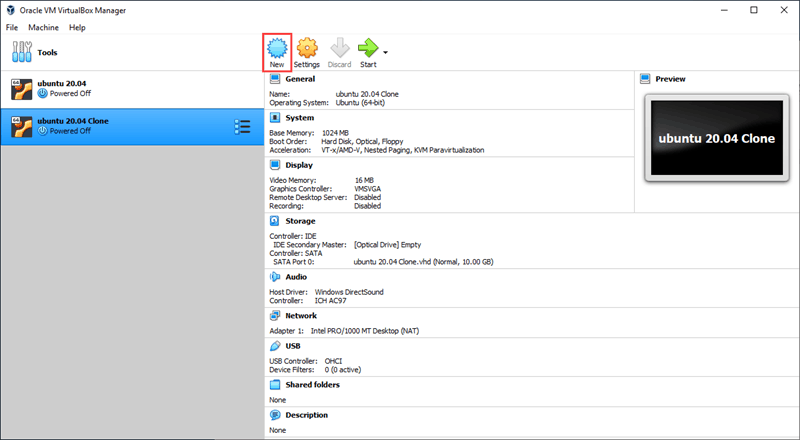


1. The install wizard might prompt you to add a VirtualBox USB device. Click **Yes** to continue.
2. Click **Finish** to complete the installation.

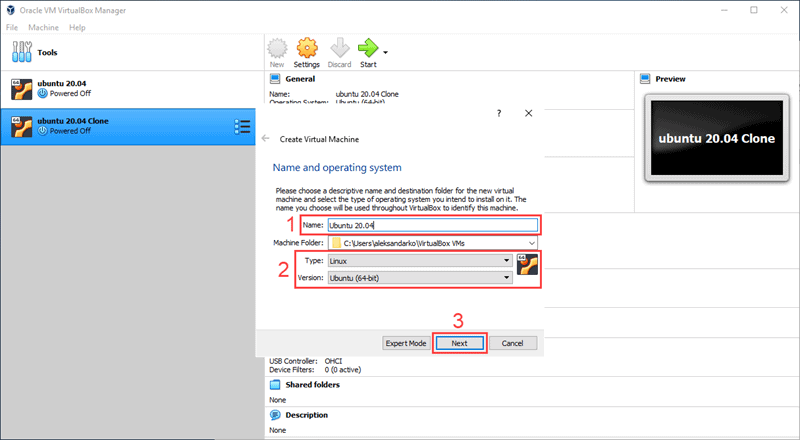


#### Step 2: Creating an Ubuntu 20.04 Virtual Box

1. Download the [Ubuntu 20.04 desktop image](https://releases.ubuntu.com/) (<https://releases.ubuntu.com/>).
2. On the VirtualBox starting screen, click **New**.

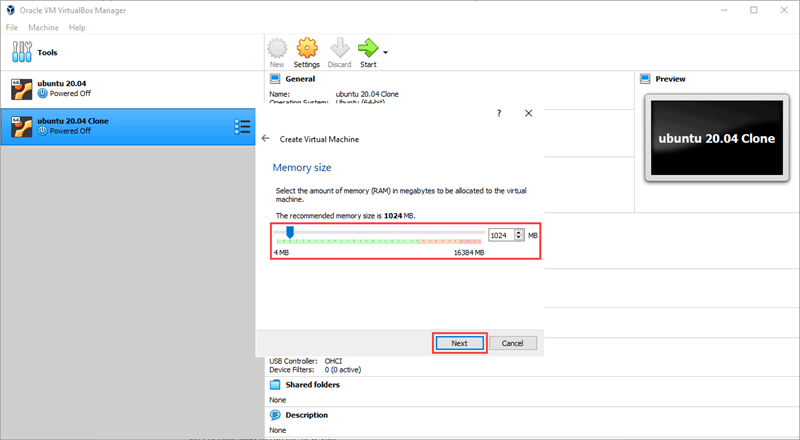


1. Type in the name of your virtual machine. If the name contains the word ‘Ubuntu’, the operating system dropdown menu will automatically update to Ubuntu (64-bit). If not, you can select the operating system manually using the dropdown menu. Click **Next** to continue.



1. Set the amount of RAM you want the virtual machine to have.

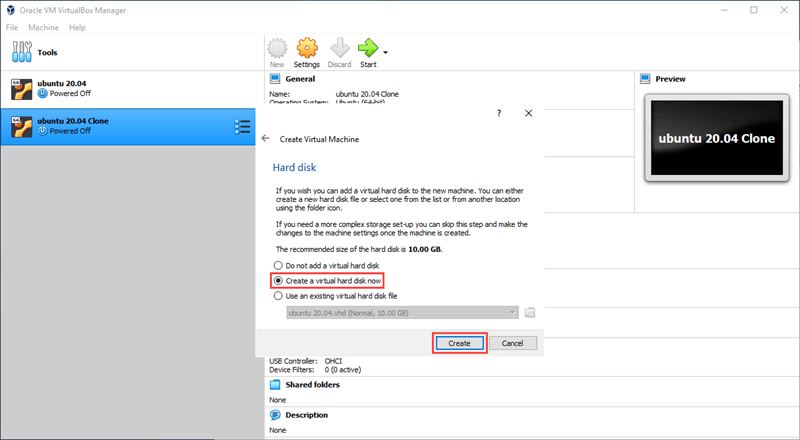
Click **Next** to continue.



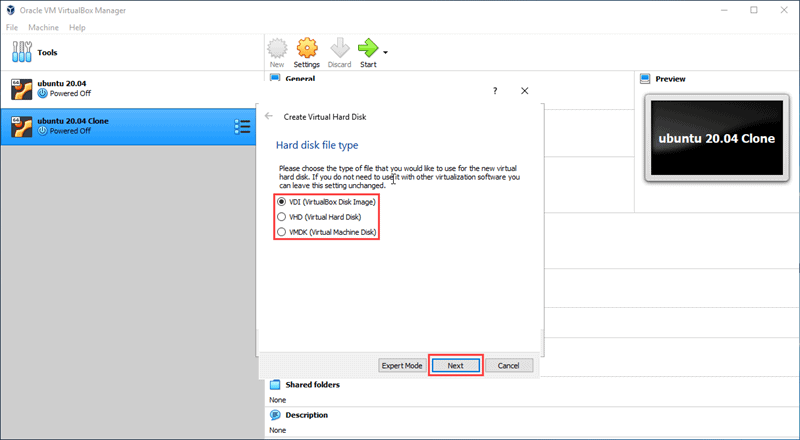
**Note:** By default, Oracle VM VirtualBox assigns 1024MB of RAM to every virtual box. Make sure you assign enough RAM to your new virtual box, or it will not work properly.

1. Select a hard disk option for your virtual machine.

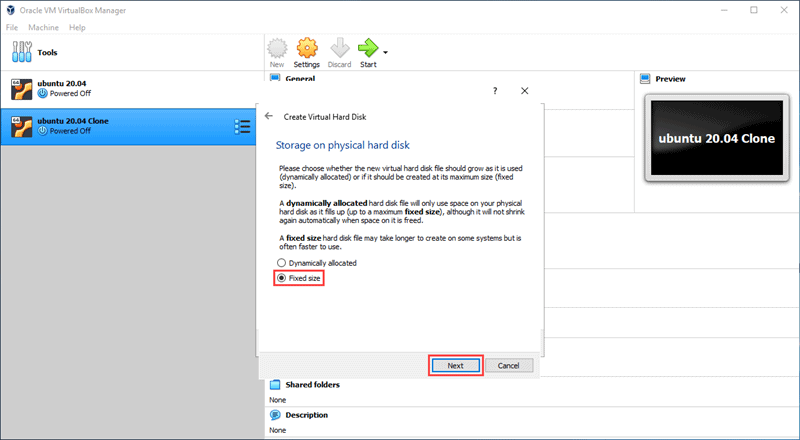
Choose **Create a virtual hard disk now** if you are creating a brand new VM. Click **Create** to continue.



1. Choose the type of virtual hard disk you want to create. Click **Next**.

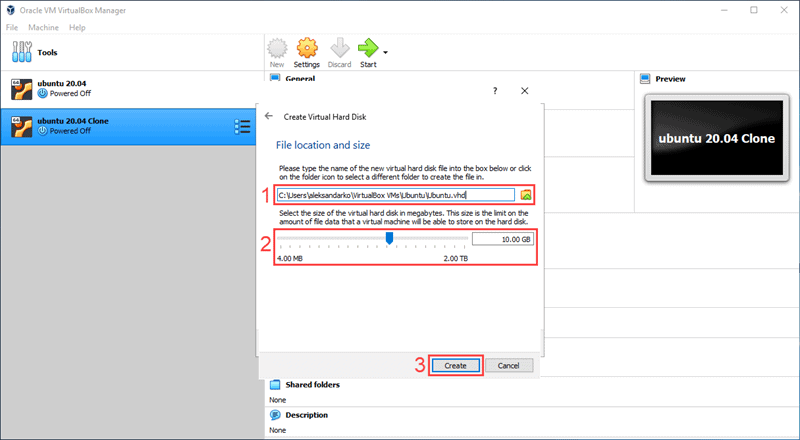


1. Choose if the new virtual hard disk has a fixed amount of space or if you want it to allocate space dynamically. Select **Fixed size** and click **Next**.



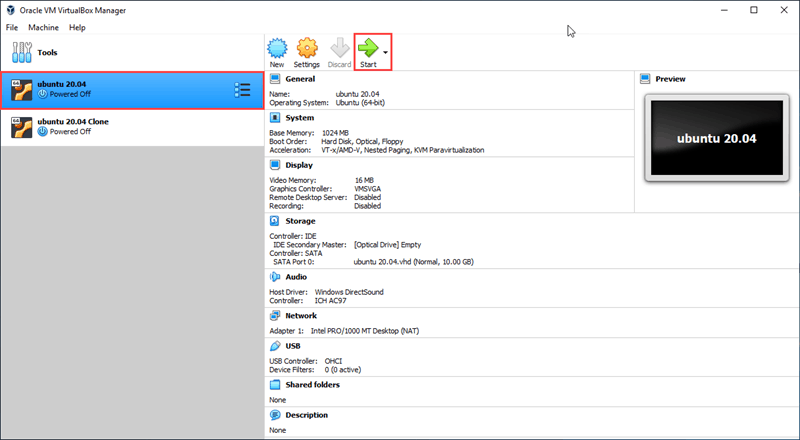
1. Select the location of the virtual hard disk on your computer and enter the size limit.

For this example, we will set the size limit to 10GB. Click **Create** to continue.



**Note:** Just like with RAM, you need to make sure you assign enough hard disk space to your virtual machine. VirtualBox sets the fixed-size hard disk at 10GB by default.

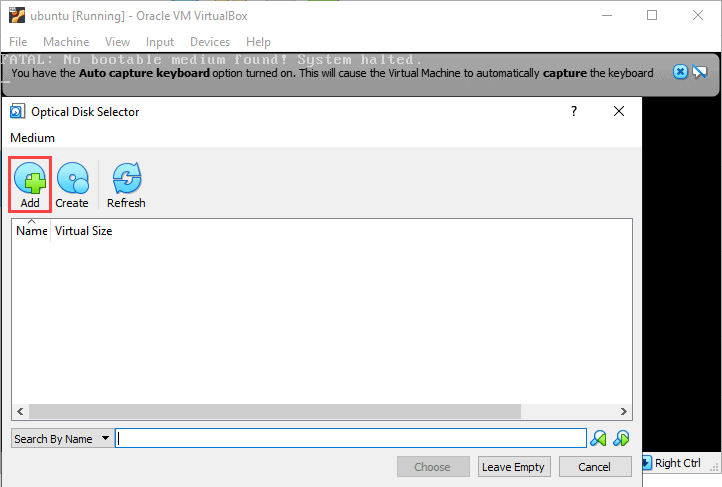
1. Select the new virtual machine you just created on the left-hand side of the VirtualBox starting screen. Click **Start** to run the new VM.



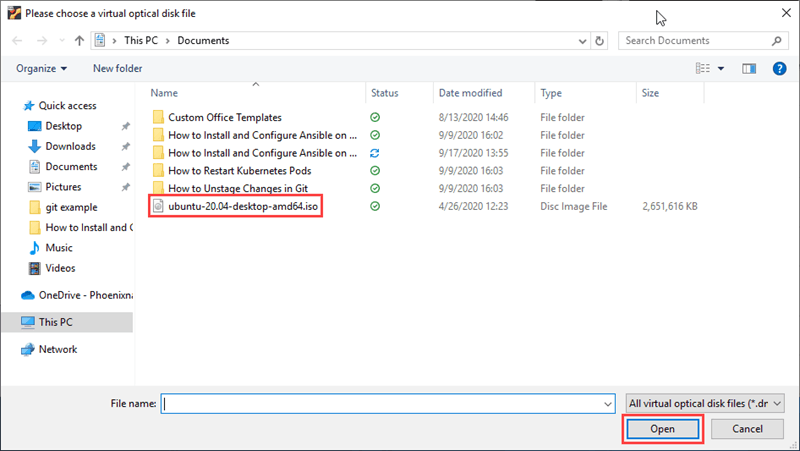
1. VirtualBox will prompt you to select a start-up disk. Click on the button next to the drop-down menu to open the optical **Disk Selector**.



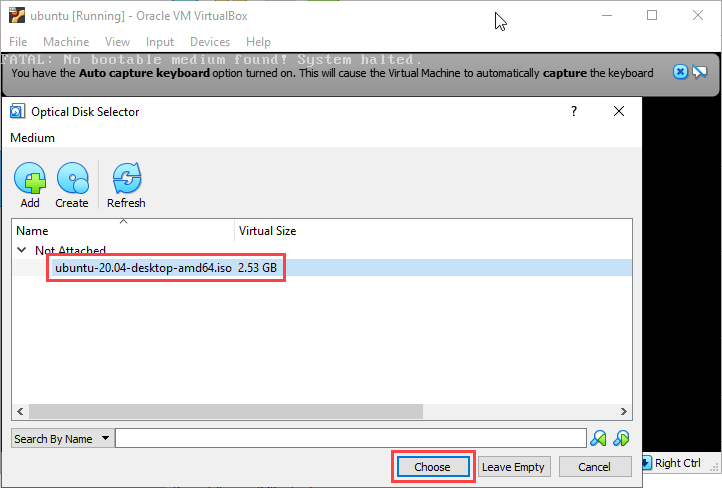
1. In the **Optical Disk Selector**, click **Add**.



1. Find the Ubuntu 20.04 desktop image you downloaded in a previous step. Select the image and click **Open**.



1. In the **Optical Disk Selector**, highlight the image you just added and click **Choose**.



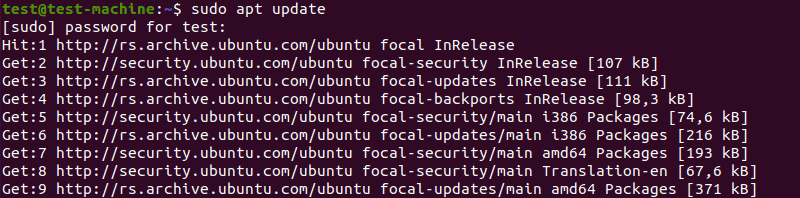
1. Click **Start** to begin the install process for Ubuntu 20.04.

#### Step 3: Installing Ansible

1. Once you are done installing Ubuntu 20.04, open the command terminal.
2. In the command terminal, enter the following command:

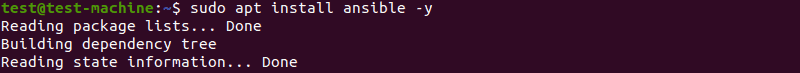
$ sudo apt update

This command refreshes the system’s package index and allows you to Install Ansible.



1. To install Ansible, use the command:

$ sudo apt install ansible -y



The rest of the install process is automated.

After you install it on you virtual machine, you can [configure Ansible](https://phoenixnap.com/kb/install-ansible-ubuntu-20-04) by setting up the [hosts inventory file](https://phoenixnap.com/kb/ansible-create-file) and checking the connections.

After running the above line of code, you are ready to manage remote machines through Ansible. Just run ansible –version to check the version and just to check whether Ansible was installed properly or not.

# YAML Basics

YAML Tutorial for DevOps: <https://youtu.be/1uFVr15xDGg>

YAML (/ˈjæməl/, rhymes with *camel*) was first proposed by Clark Evans in 2001, who designed it together with Ingy döt Netand Oren Ben-Kiki. Originally YAML was said to mean *Yet Another Markup Language*, because it was released in an era that saw a proliferation of markup languages for presentation and connectivity (HTML, XML, SGML, etc). Its initial name was intended as a tongue-in-cheek reference to the technology landscape, referencing its purpose as a markup language with the yet another construct, but it was then repurposed as *YAML Ain't Markup Language*, a recursive acronym, to distinguish its purpose as data-oriented, rather than document markup.

Ansible uses YAML syntax for expressing Ansible playbooks. This chapter provides an overview of YAML. Ansible uses YAML because it is very easy for humans to understand, read and write when compared to other data formats like XML and JSON.

Every **YAML** file optionally starts with “---” and ends with “...”.

## Understanding YAML

In this section, we will learn the different ways in which the YAML data is represented.

### key-value pair

YAML uses simple key-value pair to represent the data. The dictionary is represented in key: value pair.

**Note** − There should be space between : and value.

**Example: A student record**

--- #Optional YAML start syntax

james:

name: james john

rollNo: 34

div: B

sex: male

… #Optional YAML end syntax

#### Abbreviation

You can also use abbreviation to represent dictionaries.

**Example**

James: {name: james john, rollNo: 34, div: B, sex: male}

### Representing List

We can also represent List in YAML. Every element(member) of list should be written in a new line with same indentation starting with “- “ (- and space).

**Example**

---

countries:

- America

- China

- Canada

- Iceland

…

#### Abbreviation

You can also use abbreviation to represent lists.

**Example**

Countries: [‘America’, ‘China’, ‘Canada’, ‘Iceland’]

### List inside Dictionaries

We can use list inside dictionaries, i.e., value of key is list.

**Example**

---

james:

name: james john

rollNo: 34

div: B

sex: male

likes:

- maths

- physics

- english

…

### List of Dictionaries

We can also make list of dictionaries.

**Example**

---

- james:

name: james john

rollNo: 34

div: B

sex: male

likes:

- maths

- physics

- english

- robert:

name: robert richardson

rollNo: 53

div: B

sex: male

likes:

- biology

- chemistry

…

YAML uses “|” to include newlines while showing multiple lines and “>” to suppress newlines while showing multiple lines. Due to this we can read and edit large lines. In both the cases intendentation will be ignored.

We can also represent **Boolean** (True/false) values in YAML. where **boolean** values can be case insensitive.

**Example**

---

- james:

name: james john

rollNo: 34

div: B

sex: male

likes:

- maths

- physics

- english

result:

maths: 87

chemistry: 45

biology: 56

physics: 70

english: 80

passed: TRUE

messageIncludeNewLines: |

Congratulation!!

You passed with 79%

messageExcludeNewLines: >

Congratulation!!

You passed with 79%

## Some common words related to Ansible.

* **Service/Server** − A process on the machine that provides the service.
* **Machine** − A physical server, vm(virtual machine) or a container.
* **Target machine** − A machine we are about to configure with Ansible.
* **Task** − An action(run this, delete that) etc managed by Ansible.
* **Playbook** − The yml file where Ansible commands are written and yml is executed on a machine.

# Ad-hoc Commands

Ad hoc commands are commands which can be run individually to perform quick functions. These commands need not be performed later.

For example, you have to reboot all your company servers. For this, you will run the Adhoc commands from ‘**/usr/bin/ansible**’.

These ad-hoc commands are not used for configuration management and deployment, because these commands are of one time usage.

ansible-playbook is used for configuration management and deployment.

One of the simplest ways Ansible can be used is by using ad-hoc commands. These can be used when you want to issue some commands on a server or a bunch of servers. Ad-hoc commands are not stored for future uses but represent a fast way to interact with the desired servers.

## Parallelism and Shell Commands

Reboot your company server in 12 parallel forks at time. For this, we need to set up SSHagent for connection.

$ ssh-agent bash

$ ssh-add ~/.ssh/id\_rsa

To run reboot for all your company servers in a group, 'abc', in 12 parallel forks −

$ Ansible abc -a "/sbin/reboot" -f 12

By default, Ansible will run the above Ad-hoc commands form current user account. If you want to change this behavior, you will have to pass the username in Ad-hoc commands as follows −

$ Ansible abc -a "/sbin/reboot" -f 12 -u username

## File Transfer

You can use the Ad-hoc commands for doing **SCP** (Secure Copy Protocol) lots of files in parallel on multiple machines.

## Transferring file to many servers/machines

$ Ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"

## Creating new directory

$ Ansible abc -m file -a "dest = /path/user1/new mode = 777 owner = user1 group = user1 state = directory"

## Deleting whole directory and files

$ Ansible abc -m file -a "dest = /path/user1/new state = absent"

## Managing Packages

The Ad-hoc commands are available for yum and apt. Following are some Ad-hoc commands using yum.

The following command checks if yum package is installed or not, but does not update it.

$ Ansible abc -m yum -a "name = demo-tomcat-1 state = present"

The following command check the package is not installed.

$ Ansible abc -m yum -a "name = demo-tomcat-1 state = absent"

The following command checks the latest version of package is installed.

$ Ansible abc -m yum -a "name = demo-tomcat-1 state = latest"

## Gathering Facts

Facts can be used for implementing conditional statements in playbook. You can find adhoc information of all your facts through the following Ad-hoc command −

$ Ansible all -m setup

## More Ad-hoc Commands

For this Ansible tutorial, a simple two servers hosts file will be configured, containing host1 and host2.

### Hosts Accessible

You can make sure that the hosts are accessible from the ansible server by issuing a ping command on all hosts.

[root@ansible-server test\_ansible]# ansible -i hosts all -m ping

host1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

host2 | SUCCESS => {

"changed": false,

"ping": "pong"

}

Text

Description automatically generated

Explanation:

1. Status of the command, in this case, SUCCESS
2. Host on which the command ran
3. The command issued via the -m parameter, in this case, ping
4. With the -i parameter, you can point to the hosts file.

You can issue the same command only on a specific host if needed.

[root@ansible-server test\_ansible]# ansible -i hosts all -m ping --limit host2

host2 | SUCCESS => {

"changed": false,

"ping": "pong"

}

Text

Description automatically generated

Explanation:

1. Limit parameter can be used to issue commands only on specific hosts in the host’s file
2. Name of the host as defined in the inventory file

### Copy File

If you need to copy a file to multiple destinations rapidly, you can use the copy module in ansible which uses SCP. So the command and its output look like below:

[root@ansible-server test\_ansible]# ansible -i hosts all -m copy -a "src=/root/test\_ansible/testfile dest=/tmp/testfile"

host1 | SUCCESS => {

"changed": true,

"checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",

"dest": "/tmp/testfile",

"gid": 0,

"group": "root",

"md5sum": "d41d8cd98f00b204e9800998ecf8427e",

"mode": "0644",

"owner": "root",

"size": 0,

"src": "/root/.ansible/tmp/ansible-tmp-1562216392.43-256741011164877/source",

"state": "file",

"uid": 0

}

host2 | SUCCESS => {

"changed": true,

"checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",

"dest": "/tmp/testfile",

"gid": 0,

"group": "root",

"md5sum": "d41d8cd98f00b204e9800998ecf8427e",

"mode": "0644",

"owner": "root",

"size": 0,

"src": "/root/.ansible/tmp/ansible-tmp-1562216392.6-280302911361278/source",

"state": "file",

"uid": 0

}

Text

Description automatically generated

Explanation:

1. Copy module defined
2. Module arguments, in this case, are source absolute path and destination absolute path.
3. Ansible command output reflecting the success of the copy command and other details like the sha1 or md5 checksums for file integrity check and metadata like owner, size, or permissions.It is effortless to have a package installed on a bunch of servers. Ansible has several modules that interact with used installers, like yum, apt, dnf, etc.

### Install a package via the yum module on two Centos hosts

[root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=present'

host1 | SUCCESS => {

"changed": true,

"msg": "",

"rc": 0,

"results": [

"Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n \* base: mirror.netsite.dk\n \* elrepo: mirrors.xservers.ro\n \* epel: fedora.mirrors.telekom.ro\n \* extras: centos.mirrors.telekom.ro\n \* remi-php70: remi.schlundtech.de\n \* remi-safe: remi.schlundtech.de\n \* updates: centos.mirror.iphh.net\nResolving Dependencies\n--> Running transaction check\n---> Package ncdu.x86\_64 0:1.14-1.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================\n Package Arch Version Repository Size\n================================================================================\nInstalling:\n ncdu x86\_64 1.14-1.el7 epel 51 k\n\nTransaction Summary\n================================================================================\nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86\_64 1/1 \n Verifying : ncdu-1.14-1.el7.x86\_64 1/1 \n\nInstalled:\n ncdu.x86\_64 0:1.14-1.el7 \n\nComplete!\n"

]

}

host2 | SUCCESS => {

"changed": true,

"msg": "",

"rc": 0,

"results": [

"Loaded plugins: fastestmirror\nLoading mirror speeds from cached hostfile\n \* base: mirror.netsite.dk\n \* elrepo: mirrors.leadhosts.com\n \* epel: mirrors.nav.ro\n \* extras: centos.mirrors.telekom.ro\n \* remi-php70: mirrors.uni-ruse.bg\n \* remi-safe: mirrors.uni-ruse.bg\n \* updates: centos.mirror.iphh.net\nResolving Dependencies\n--> Running transaction check\n---> Package ncdu.x86\_64 0:1.14-1.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================\n Package Arch Version Repository Size\n================================================================================\nInstalling:\n ncdu x86\_64 1.14-1.el7 epel 51 k\n\nTransaction Summary\n================================================================================\nInstall 1 Package\n\nTotal download size: 51 k\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Installing : ncdu-1.14-1.el7.x86\_64 1/1 \n Verifying : ncdu-1.14-1.el7.x86\_64 1/1 \n\nInstalled:\n ncdu.x86\_64 0:1.14-1.el7 \n\nComplete!\n"

]

}

Text

Description automatically generated

Explanation:

1. Yum module is used in this example
2. It defines the module arguments, and in this case, you will choose the name of the package and its state. If the state is absent, for example, the package will be searched and if found, removed
3. When colored in yellow, you will see the output of the ansible command with the state changed, meaning in this case, that the package was found and installed.
4. Status of the yum install command issued via ansible. In this case the package ncdu.x86\_64 0:1.14-1.el7 was installed.

Of course, all of the yum installer options can be used via ansible, including update, install, latest version, or remove.

### Remove the previously installed ncdu package

[root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=absent'

host1 | SUCCESS => {

"changed": true,

"msg": "",

"rc": 0,

"results": [

"Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n---> Package ncdu.x86\_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================\n Package Arch Version Repository Size\n================================================================================\nRemoving:\n ncdu x86\_64 1.14-1.el7 @epel 87 k\n\nTransaction Summary\n================================================================================\nRemove 1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Erasing : ncdu-1.14-1.el7.x86\_64 1/1 \n Verifying : ncdu-1.14-1.el7.x86\_64 1/1 \n\nRemoved:\n ncdu.x86\_64 0:1.14-1.el7 \n\nComplete!\n"

]

}

host2 | SUCCESS => {

"changed": true,

"msg": "",

"rc": 0,

"results": [

"Loaded plugins: fastestmirror\nResolving Dependencies\n--> Running transaction check\n---> Package ncdu.x86\_64 0:1.14-1.el7 will be erased\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================================\n Package Arch Version Repository Size\n================================================================================\nRemoving:\n ncdu x86\_64 1.14-1.el7 @epel 87 k\n\nTransaction Summary\n================================================================================\nRemove 1 Package\n\nInstalled size: 87 k\nDownloading packages:\nRunning transaction check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Erasing : ncdu-1.14-1.el7.x86\_64 1/1 \n Verifying : ncdu-1.14-1.el7.x86\_64 1/1 \n\nRemoved:\n ncdu.x86\_64 0:1.14-1.el7 \n\nComplete!\n"

]

}

Text

Description automatically generated

Explanation:

1. The output of the yum command shows that the package was removed.

### Gather some facts about the system

Another useful and essential feature that ansible uses to interact with the client’s server is to gather some facts about the system. So, it fetches hardware, software, and versioning information from the system and stores each value in a variable that can be later on used.

If you need detailed information about the systems to be modified via ansible, the next command can be used. The setup module gathers facts from the system variables.

[root@ansible-server test\_ansible]# ansible -i hosts all -m setup

### Dry-run p4.yml playbook

[root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml --check

### Run p4.yml playbook with password authentication for all hosts

[root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml -k

# Ansible Playbooks

**Ansible Playbooks** are the way of sending commands to remote systems through scripts. Ansible playbooks are used to configure complex system environments to increase flexibility by executing a script to one or more systems. Ansible playbooks tend to be more of a configuration language than a programming language.

Ansible playbook commands use YAML format, so there is not much syntax needed, but indentation must be respected. Like the name is saying, a playbook is a collection of plays. Through a playbook, you can designate specific roles to some hosts and other roles to other hosts. By doing so, you can orchestrate multiple servers in very diverse scenarios, all in one playbook.

Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. YAML stands for Yet Another Markup Language. **Playbooks** are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks.

Playbooks contain the steps which the user wants to execute on a particular machine. Playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.

## Playbook Structure

Each playbook is an aggregation of one or more plays in it. Playbooks are structured using Plays. There can be more than one play inside a playbook.

The function of a play is to map a set of instructions defined against a particular host.

YAML is a strict typed language; so, extra care needs to be taken while writing the YAML files. There are different YAML editors but we will prefer to use a simple editor like notepad++. Just open notepad++ and copy and paste the below yaml and change the language to YAML (Language → YAML).

A YAML starts with --- (3 hyphens)

## Create a Playbook

Let us start by writing a sample YAML file. We will walk through each section written in a yaml file.

---

name: install and configure DB

hosts: testServer

become: yes

vars:

oracle\_db\_port\_value : 1521

tasks:

-name: Install the Oracle DB

yum: <code to install the DB>

-name: Ensure the installed service is enabled and running

service:

name: <your service name>

The above is a sample Playbook where we are trying to cover the basic syntax of a playbook. Save the above content in a file as **test.yml**. A YAML syntax needs to follow the correct indentation and one needs to be a little careful while writing the syntax.

## The Different YAML Tags

Let us now go through the different YAML tags. The different tags are described below:

### name

This tag specifies the name of the Ansible playbook. As in what this playbook will be doing. Any logical name can be given to the playbook.

### hosts

This tag specifies the lists of hosts or host group against which we want to run the task. The hosts field/tag is mandatory. It tells Ansible on which hosts to run the listed tasks. The tasks can be run on the same machine or on a remote machine. One can run the tasks on multiple machines and hence hosts tag can have a group of hosts’ entry as well.

### vars

Vars tag lets you define the variables which you can use in your playbook. Usage is similar to variables in any programming language.

### tasks

All playbooks should contain tasks or a list of tasks to be executed. Tasks are a list of actions one needs to perform. A tasks field contains the name of the task. This works as the help text for the user. It is not mandatory but proves useful in debugging the playbook. Each task internally links to a piece of code called a module. A module that should be executed, and arguments that are required for the module you want to execute.

## Examples

To have all the details precise before continuing with Ansible playbook examples, we must first define a task. These are the interface to ansible modules for roles and playbooks.

### One playbook with one play

Now, let’s learn Ansible playbook through an example with one playbook with one play, containing multiple tasks as below:

---

- hosts: group1

tasks:

- name: Install lldpad package

yum:

name: lldpad

state: latest

- name: check lldpad service status

service:

name: lldpad

state: started

Text

Description automatically generated

In the above Ansible playbook example, the group1 of hosts in the host’s file is targeted for lldpad package installation using the yum module and afterward the service lldpad created after the installation is then started using the service module used mostly to interact with systemd ensemble.

Explanation:

1. Group of hosts on which the playbook will run
2. Yum module is used in this task for lldpad installation
3. The service module is used to check if the service is up and running after installation

Each ansible playbook works with an inventory file. The inventory file contains a list of servers divided into groups for better control for details like IP Address and SSH port for each host.

The inventory file you can use for this Ansible playbook example looks like below. There are two groups, named group1 and group2 each containing host1 and host2 respectively.

[group1]

host1 ansible\_host=192.168.100.2 ansible\_ssh\_port=22

[group2]

host2 ansible\_host=192.168.100.3 ansible\_ssh\_port=22

Graphical user interface, text, application

Description automatically generated

Explanation:

1. Group name
2. Hostname, with IP address and ssh port, in this case, the default one, 22.

### Two plays for two hosts

Another useful Ansible playbook example containing this time two plays for two hosts is the next one. For the first group of hosts, group1, selinux will be enabled. If it is enabled, then a message will appear on the screen of the host.

For the second group of hosts, httpd package will be installed only if the ansible\_os\_family is RedHat and ansible\_system\_vendor is HP.

Ansible\_os\_family and ansible\_system\_vendor are variables gathered with gather\_facts option and can be used like in this conditional example.

---

- hosts: group1

tasks:

- name: Enable SELinux

selinux:

state: enabled

when: ansible\_os\_family == 'Debian'

register: enable\_selinux

- debug:

Imsg: "Selinux Enabled. Please restart the server to apply changes."

when: enable\_selinux.changed == true

- hosts: group2

tasks:

- name: Install apache

yum:

name: httpd

state: present

when: ansible\_system\_vendor == 'HP' and ansible\_os\_family == 'RedHat'

A picture containing text

Description automatically generated

Explanation:

1. Example of the when clause, In this case, when OS type is Debian. The ansible\_os\_family variable is gathered via gather\_facts functionality.
2. The task output is registered for future use, with its name enable\_selinux
3. Another example of the when clause. In this case, a message will be displayed for the host user if the SELinux was indeed enabled before.
4. Another example of the when clause consisting of two rules

Besides tasks, there are also some particular tasks called handlers. Handlers must have a unique name throughout the playbook. These work in the same way as a regular task but a handler can be notified via a notifier.

If a handler is not notified during the run of the playbook, it will not run. However, if more than one task notifies a handler, this will run only once after all the tasks are finished.

### Task Calls Upon Another Task

In the example shown below, you can see how a specific task has a notify section which calls upon another task. If the output of the first task is changed, then a handler task will be called. The best example is to have a configuration file changed and afterward restart that specific service.

---

- hosts: group2

tasks:

- name: sshd config file modify port

lineinfile:

path: /etc/ssh/sshd\_config

regexp: 'Port 28675'

line: '#Port 22'

notify:

- restart sshd

handlers

- name: restart sshd

service: sshd

name: sshd

state: restarted

In this case, if the first task, “sshd config file modify port” is changed, meaning that if the port is not 28675 in the first place, then it will be modified and the task will notify the handler with the same name to run, and it will restart the sshd service.

Text

Description automatically generated

Explanation:

1. Example of a notifier
2. Example of a handler

# Ansible Roles

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Each role is basically limited to a particular functionality or desired output, with all the necessary steps to provide that result either within that role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

When dealing with extensive playbooks, it is easier to split the tasks into roles. This also helps in reusing the roles in the future. Roles are a collection of tasks, which can be moved from one playbook to another, can be run independently but only through a playbook file.

Roles are stored in separate directories and have a particular directory structure.

[root@ansible-server test2]# tree

.

`-- role1

|-- defaults

| `-- main.yml

|-- handlers

| `-- main.yml

|-- meta

| `-- main.yml

|-- README.md

|-- tasks

| `-- main.yml

|-- tests

| |-- inventory

| `-- test.yml

`-- vars

`-- main.yml

7 directories, 8 files

The yaml file in the defaults directory contains a list of default variables that are to be used along with the playbook. The handlers directory is used to store handlers. The meta-directory is supposed to have information about the author and role dependencies. In the tasks directory, there is the main yaml file for the role.

The tests directory contains a sample yaml playbook file and a sample inventory file and is mostly used for testing purposes before creating the actual role.

The vars directory contains the yaml file in which all the variables used by the role will be defined. The directory templates and the directory files should contain files and templates that will be used by the tasks in the role.

## Creating a New Role

The directory structure for roles is essential to create a new role.

### Role Structure

Roles have a structured layout on the file system. The default structure can be changed but for now let us stick to defaults (*given above*).

Each role is a directory tree in itself. The role name is the directory name within the /roles directory.

$ ansible-galaxy -h

### Usage

ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup] [--help] [options] ...

### Options

* **-h, --help** − Show this help message and exit.
* **-v, --verbose** − Verbose mode (-vvv for more, -vvvv to enable connection debugging)
* **--version** − Show program's version number and exit.

## Creating a Role Directory

To create the directory tree for a role, you should use the following command with the last parameter, the role name:

[root@ansible-server test2]# ansible-galaxy init <role-dir-name>

Let’s take an example:

$ ansible-galaxy init ajsrole

ERROR! The API server (https://galaxy.ansible.com/api/) is not responding, please try again later.

$ ansible-galaxy init --force --offline ajsrole

- ajsrole was created successfully

$ tree ajsrole/

ajsrole/

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── README.md

├── tasks

│ └── main.yml

├── templates

├── tests

│ ├── inventory

│ └── test.yml

└── vars

└── main.yml

8 directories, 8 files

Not all the directories will be used in the example and we will show the use of some of them in the example.

## Utilizing Roles in Playbook

This is the code of the playbook we have written for demo purpose. This code is of the playbook ajs\_orchestrate.yml. We have defined the hosts: **tomcat-node** and called the two roles – **install-tomcat** and **start-tomcat**.

The problem statement is that we have a war which we need to deploy on a machine via Ansible.

---

- hosts: tomcat-node

roles:

- {role: install-tomcat}

- {role: start-tomcat}

Contents of our directory structure from where we are running the playbook.



$ ls

ansible.cfg hosts roles ajs\_orchestrate.retry ajs\_orchestrate.yml



There is a tasks directory under each directory and it contains a main.yml. The main.yml contents of install-tomcat are −

---

#Install ajs artifacts

-

block:

- name: Install Tomcat artifacts

action: >

yum name = "demo-tomcat-1" state = present

register: Output

always:

- debug:

msg:

- "Install Tomcat artifacts task ended with message: {{Output}}"

- "Installed Tomcat artifacts - {{Output.changed}}"

The contents of main.yml of the start tomcat are −

#Start Tomcat

-

block:

- name: Start Tomcat

command: <path of tomcat>/bin/startup.sh"

register: output

become: true

always:

- debug:

msg:

- "Start Tomcat task ended with message: {{output}}"

- "Tomcat started - {{output.changed}}"

The advantage of breaking the playbook into roles is that anyone who wants to use the Install tomcat feature can call the Install Tomcat role.

## Breaking a Playbook into a Role

If not for the roles, the content of the main.yml of the respective role can be copied in the playbook **yml** file. But to have modularity, roles were created.

Any logical entity which can be reused as a reusable function, that entity can be moved to role. The example for this is shown above

Run the command to run the playbook.

-vvv option for verbose output – verbose output

$ cd ajs-playbook/

This is the command to run the playbook

$ sudo ansible-playbook -i hosts ajs\_orchestrate.yml –vvv

-----------------------------------------------------------------

### Output

The generated output is as seen on the screen −

Using **/users/demo/ajs-playbook/ansible.cfg** as config file.

PLAYBOOK: ajs\_orchestrate.yml \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 plays in ajs\_orchestrate.yml

PLAY [tomcat-node] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tuesday 21 November 2017 13:02:05 +0530 (0:00:00.056) 0:00:00.056 \*\*\*\*\*\*

Using module file /usr/lib/python2.7/sitepackages/ansible/modules/system/setup.py

<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root

<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'

<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo

/root/.ansible/tmp/ansible-tmp-1511249525.88-259535494116870 `" &&

echo ansible-tmp-1511249525.88-259535494116870="`

echo /root/.ansible/tmp/ansibletmp-1511249525.88-259535494116870 `" ) && sleep 0'

<localhost> PUT /tmp/tmpPEPrkd TO

/root/.ansible/tmp/ansible-tmp-1511249525.88259535494116870/setup.py

<localhost> EXEC /bin/sh -c 'chmod u+x

/root/.ansible/tmp/ansible-tmp1511249525.88-259535494116870/

/root/.ansible/tmp/ansible-tmp-1511249525.88259535494116870/setup.py && sleep 0'

<localhost> EXEC /bin/sh -c '/usr/bin/python

/root/.ansible/tmp/ansible-tmp1511249525.88-259535494116870/setup.py; rm -rf

"/root/.ansible/tmp/ansible-tmp1511249525.88-259535494116870/" > /dev/null 2>&1 && sleep 0'

ok: [server1]

META: ran handlers

TASK [install-tomcat : Install Tomcat artifacts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:5

Tuesday 21 November 2017 13:02:07 +0530 (0:00:01.515) 0:00:01.572 \*\*\*\*\*\*

Using module file /usr/lib/python2.7/sitepackages/ansible/modules/packaging/os/yum.py

<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root

<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'

<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo

/root/.ansible/tmp/ansible-tmp-1511249527.34-40247177825302 `" && echo

ansibletmp-1511249527.34-40247177825302="` echo

/root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302 `" ) && sleep 0'

<localhost> PUT /tmp/tmpu83chg TO

/root/.ansible/tmp/ansible-tmp-1511249527.3440247177825302/yum.py

<localhost> EXEC /bin/sh -c 'chmod u+x

/root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302/

/root/.ansible/tmp/ansible-tmp-1511249527.3440247177825302/yum.py && sleep 0'

<localhost> EXEC /bin/sh -c '/usr/bin/python

/root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302/yum.py; rm -rf

"/root/.ansible/tmp/ansible-tmp1511249527.34-40247177825302/" > /dev/null 2>

&1 && sleep 0'

changed: [server1] => {

"changed": true,

"invocation": {

"module\_args": {

"conf\_file": null,

"disable\_gpg\_check": false,

"disablerepo": null,

"enablerepo": null,

"exclude": null,

"install\_repoquery": true,

"installroot": "/",

"list": null,

"name": ["demo-tomcat-1"],

"skip\_broken": false,

"state": "present",

"update\_cache": false,

"validate\_certs": true

}

},

"msg": "",

"rc": 0,

"results": [

"Loaded plugins: product-id,

search-disabled-repos,

subscriptionmanager\nThis system is not registered to Red Hat Subscription Management.

You can use subscription-manager to register.\nResolving Dependencies\n-->

Running transaction check\n--->

Package demo-tomcat-1.noarch 0:SNAPSHOT-1 will be installed\n--> Finished Dependency

Resolution\n\nDependencies Resolved\n

\n================================================================================\n

Package Arch Version Repository

Size\n==================================================================\nInstalling:\n

demo-tomcat-1 noarch SNAPSHOT-1 demo-repo1 7.1 M\n\nTransaction

Summary\n==================================================================\nInstall 1

Package\n\nTotal download size: 7.1 M\nInstalled size: 7.9 M\nDownloading

packages:\nRunning transaction

check\nRunning transaction test\nTransaction test succeeded\nRunning transaction\n Installing :

demotomcat-1-SNAPSHOT-1.noarch 1/1 \n Verifying :

demo-tomcat-1-SNAPSHOT-1.noarch 1/1 \n\nInstalled:\n

demo-tomcat-1.noarch 0:SNAPSHOT-1 \n\nComplete!\n"

]

}

TASK [install-tomcat : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:11

Tuesday 21 November 2017 13:02:13 +0530 (0:00:06.757) 0:00:08.329 \*\*\*\*\*\*

ok: [server1] => {

"changed": false,

"msg": [

"Install Tomcat artifacts task ended with message: {

u'msg': u'', u'changed': True, u'results':

[u'Loaded plugins: product-id,

search-disabledrepos,

subscription-manager\\nThis system is not registered to Red Hat Subscription Management.

You can use subscription-manager to register.\\nResolving Dependencies\\n-->

Running transaction check\\n--->

Package demo-tomcat-1.noarch 0:SNAPSHOT-1 will be installed\\n-->

Finished Dependency Resolution\\n

\\nDependencies

Resolved\\n\\n==================================================================\\n

Package Arch Version Repository

Size\\n========================================================================

=====\\nInstalling:\\n demo-tomcat-1 noarch SNAPSHOT-1 demo-repo1 7.1 M\\n\\nTransaction

Summary\\n=========================================================\\nInstall 1

Package\\n\\nTotal download size: 7.1 M\\nInstalled size: 7.9 M\\nDownloading

packages:\\nRunning

transaction check\\nRunning transaction test\\nTransaction test succeeded\\nRunning

transaction\\n

Installing : demo-tomcat-1-SNAPSHOT-1.noarch 1/1 \\n Verifying :

demo-tomcat-1-SNAPSHOT-1.noarch

1/1 \\n\\nInstalled:\\n demo-tomcat-1.noarch 0:SNAPSHOT-1 \\n\\nComplete!\\n'], u'rc': 0

}",

"Installed Tomcat artifacts - True"

]

}

TASK [install-tomcat : Clean DEMO environment] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:19

Tuesday 21 November 2017 13:02:13 +0530 (0:00:00.057) 0:00:08.387 \*\*\*\*\*\*

[WARNING]: when statements should not include jinja2 templating delimiters such as {{ }} or

{% %}. Found: {{installationOutput.changed}}

Using module file /usr/lib/python2.7/sitepackages/ansible/modules/files/file.py

<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root

<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'

<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo

/root/.ansible/tmp/ansible-tmp-1511249534.13-128345805983963 `" && echo

ansible-tmp-1511249534.13-128345805983963="` echo

/root/.ansible/tmp/ansibletmp-1511249534.13-128345805983963 `" ) && sleep 0'

<localhost> PUT /tmp/tmp0aXel7 TO

/root/.ansible/tmp/ansible-tmp-1511249534.13128345805983963/file.py

<localhost> EXEC /bin/sh -c 'chmod u+x

/root/.ansible/tmp/ansible-tmp1511249534.13-128345805983963/

/root/.ansible/tmp/ansible-tmp-1511249534.13128345805983963/file.py && sleep 0'

<localhost> EXEC /bin/sh -c '/usr/bin/python

/root/.ansible/tmp/ansible-tmp1511249534.13-128345805983963/file.py; rm -rf

"/root/.ansible/tmp/ansible-tmp1511249534.13-128345805983963/" > /dev/null 2>&1

&& sleep 0'

changed: [server1] => {

"changed": true,

"diff": {

"after": {

"path": "/users/demo/DEMO",

"state": "absent"

},

"before": {

"path": "/users/demo/DEMO",

"state": "directory"

}

},

"invocation": {

"module\_args": {

"attributes": null,

"backup": null,

"content": null,

"delimiter": null,

"diff\_peek": null,

"directory\_mode": null,

"follow": false,

"force": false,

"group": null,

"mode": null,

"original\_basename": null,

"owner": null,

"path": "/users/demo/DEMO",

"recurse": false,

"regexp": null,

"remote\_src": null,

"selevel": null,

"serole": null,

"setype": null,

"seuser": null,

"src": null,

"state": "absent",

"unsafe\_writes": null,

"validate": null

}

},

"path": "/users/demo/DEMO",

"state": "absent"

}

TASK [install-tomcat : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:29

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.257) 0:00:08.645 \*\*\*\*\*\*

ok: [server1] => {

"changed": false,

"msg": [

"Clean DEMO environment task ended with message:{u'diff': {u'after': {u'path':

u'/users/demo/DEMO', u'state': u'absent'},

u'before': {u'path': u'/users/demo/DEMO', u'state': u'directory'}}, u'state': u'absent',

u'changed': True, u'path': u'/users/demo/DEMO'}",

"check value :True"

]

}

TASK [install-tomcat : Copy Tomcat to user home] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:37

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.055) 0:00:08.701 \*\*\*\*\*\*

[WARNING]: when statements should not include jinja2 templating delimiters such as {{ }} or

{% %}. Found: {{installationOutput.changed}}

Using module file /usr/lib/python2.7/sitepackages/ansible/modules/commands/command.py

<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root

<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'

<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo

/root/.ansible/tmp/ansible-tmp-1511249534.43-41077200718443 `" && echo

ansibletmp-1511249534.43-41077200718443="` echo

/root/.ansible/tmp/ansible-tmp1511249534.43-41077200718443 `" ) && sleep 0'

<localhost> PUT /tmp/tmp25deWs TO

/root/.ansible/tmp/ansible-tmp-1511249534.4341077200718443/command.py

<localhost> EXEC /bin/sh -c 'chmod u+x

/root/.ansible/tmp/ansible-tmp1511249534.43-41077200718443/

/root/.ansible/tmp/ansible-tmp-1511249534.4341077200718443/command.py && sleep 0'

<localhost> EXEC /bin/sh -c '/usr/bin/python

/root/.ansible/tmp/ansible-tmp1511249534.43-41077200718443/command.py; rm -rf

"/root/.ansible/tmp/ansibletmp-1511249534.43-41077200718443/" > /dev/null 2>&1

&& sleep 0'

changed: [server1] => {

"changed": true,

"cmd": [

"cp",

"-r",

"/opt/ansible/tomcat/demo",

"/users/demo/DEMO/"

],

"delta": "0:00:00.017923",

"end": "2017-11-21 13:02:14.547633",

"invocation": {

"module\_args": {

"\_raw\_params": "cp -r /opt/ansible/tomcat/demo /users/demo/DEMO/",

"\_uses\_shell": false,

"chdir": null,

"creates": null,

"executable": null,

"removes": null,

"warn": true

}

},

"rc": 0,

"start": "2017-11-21 13:02:14.529710",

"stderr": "",

"stderr\_lines": [],

"stdout": "",

"stdout\_lines": []

}

TASK [install-tomcat : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/install-tomcat/tasks/main.yml:47

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.260) 0:00:08.961 \*\*\*\*\*\*

ok: [server1] => {

"changed": false,

"msg": "Copy Tomcat to user home task ended with message {

'stderr\_lines': [], u'changed': True, u'end': u'2017-11-21 13:02:14.547633', u'stdout':

u'', u'cmd': [u'cp', u'-r', u'/opt/ansible/tomcat/demo', u'/users/demo/DEMO/'], u'rc': 0,

u'start': u'2017-11-21 13:02:14.529710', u'stderr': u'', u'delta': u'0:00:00.017923',

'stdout\_lines': []}"

}

TASK [start-tomcat : Start Tomcat] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/start-tomcat/tasks/main.yml:5

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.044) 0:00:09.006 \*\*\*\*\*\*

Using module file /usr/lib/python2.7/sitepackages/ansible/modules/commands/command.py

<localhost> ESTABLISH LOCAL CONNECTION FOR USER: root

<localhost> EXEC /bin/sh -c 'echo ~ && sleep 0'

<localhost> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo

/root/.ansible/tmp/ansible-tmp-1511249534.63-46501211251197 `" && echo

ansibletmp-1511249534.63-46501211251197="` echo

/root/.ansible/tmp/ansible-tmp1511249534.63-46501211251197 `" ) && sleep 0'

<localhost> PUT /tmp/tmp9f06MQ TO

/root/.ansible/tmp/ansible-tmp-1511249534.6346501211251197/command.py

<localhost> EXEC /bin/sh -c 'chmod u+x

/root/.ansible/tmp/ansible-tmp1511249534.63-46501211251197/

/root/.ansible/tmp/ansible-tmp-1511249534.6346501211251197/command.py && sleep 0'

<localhost> EXEC /bin/sh -c '/usr/bin/python

/root/.ansible/tmp/ansible-tmp1511249534.63-46501211251197/command.py; rm -rf

"/root/.ansible/tmp/ansibletmp-1511249534.63-46501211251197/" > /dev/null 2>&1

&& sleep 0'

changed: [server1] => {

"changed": true,

"cmd": [ "/users/demo/DEMO/bin/startup.sh" ],

"delta": "0:00:00.020024",

"end": "2017-11-21 13:02:14.741649",

"invocation": {

"module\_args": {

"\_raw\_params": "/users/demo/DEMO/bin/startup.sh",

"\_uses\_shell": false,

"chdir": null,

"creates": null,

"executable": null,

"removes": null,

"warn": true

}

},

"rc": 0,

"start": "2017-11-21 13:02:14.721625",

"stderr": "",

"stderr\_lines": [],

"stdout": "Tomcat started.",

"stdout\_lines": [ "Tomcat started." ]

}

TASK [start-tomcat : debug] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

task path: /users/demo/ ajs-playbook/roles/start-tomcat/tasks/main.yml:10

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.150) 0:00:09.156 \*\*\*\*\*\*

ok: [server1] => {

"changed": false,

"msg": [

"Start Tomcat task ended with message: {'

stderr\_lines': [], u'changed': True, u'end': u'2017-11-21 13:02:14.741649', u'stdout':

u'Tomcat started.', u'cmd': [u'/users/demo/DEMO/bin/startup.sh'], u'rc': 0, u'start':

u'2017-11-21 13:02:14.721625', u'stderr': u'', u'delta': u'0:00:00.020024',

'stdout\_lines': [u'Tomcat started.']}",

"Tomcat started - True"

]

}

META: ran handlers

META: ran handlers

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server1 : ok = 9 changed = 4 unreachable = 0 failed = 0

Tuesday 21 November 2017 13:02:14 +0530 (0:00:00.042) 0:00:09.198 \*\*\*\*\*\*

===============================================================================

install-tomcat : Install Tomcat artifacts ------------------------------- 6.76s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:5 --------------

Gathering Facts --------------------------------------------------------- 1.52s

------------------------------------------------------------------------------

install-tomcat : Copy Tomcat to user home ------------------------------- 0.26s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:37 -------------

install-tomcat : Clean DEMO environment --------------------------------- 0.26s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:19 -------------

start-tomcat : Start Tomcat --------------------------------------------- 0.15s

/users/demo/ajs-playbook/roles/start-tomcat/tasks/main.yml:5 ----------------

install-tomcat : debug -------------------------------------------------- 0.06s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:11 -------------

install-tomcat : debug -------------------------------------------------- 0.06s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:29 -------------

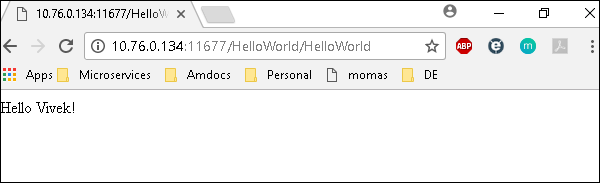
install-tomcat : debug -------------------------------------------------- 0.04s

/users/demo/ajs-playbook/roles/install-tomcat/tasks/main.yml:47 -------------

start-tomcat : debug ---------------------------------------------------- 0.04s

/users/demo/ajs-playbook/roles/start-tomcat/tasks/main.yml:10 ---------------

Hit the following URL and you will be directed to a page as shown below − **http://10.76.0.134:11677/HelloWorld/HelloWorld**



The deployed war just has a servlet which displays “Hello World”. The detailed output shows the time taken by each and every task because of the entry added in ansible.cfg file −

[defaults]

callback\_whitelist = profile\_tasks

# Ansible Variables

Variable in playbooks are **very similar** to using variables in any programming language. It helps you to use and assign a value to a variable and use that anywhere in the playbook. One can put conditions around the value of the variables and accordingly use them in the playbook.

**Example**

- hosts : <your hosts>

vars:

tomcat\_port : 8080

In the above example, we have defined a variable name **tomcat\_port** and assigned the value 8080 to that variable and can use that in your playbook wherever needed.

Now taking a reference from the example shared. The following code is from one of the roles (install-tomcat):

block:

- name: Install Tomcat artifacts

action: >

yum name = "demo-tomcat-1" state = present

register: Output

always:

- debug:

msg:

- "Install Tomcat artifacts task ended with message: {{Output}}"

- "Installed Tomcat artifacts - {{Output.changed}}"

Here, the output is the variable used.

Let us walk through all the keywords used in the above code −

* **block** − Ansible syntax to execute a given block.
* **name** − Relevant name of the block - this is used in logging and helps in debugging that which all blocks were successfully executed.
* **action** − The code next to action tag is the task to be executed. The action again is a Ansible keyword used in yaml.
* **register** − The output of the action is registered using the register keyword and Output is the variable name which holds the action output.
* **always** − Again a Ansible keyword , it states that below will always be executed.
* **msg** − Displays the message.

## Usage of variable - {{Output}}

This will read the value of variable Output. Also as it is used in the msg tab, it will print the value of the output variable.

Additionally, you can use the sub properties of the variable as well. Like in the case checking {{Output.changed}} whether the output got changed and accordingly use it.

## Exception Handling in Playbooks

Exception handling in Ansible is similar to exception handling in any programming language. An example of the exception handling in playbook is shown below.

tasks:

- name: Name of the task to be executed

block:

- debug: msg = 'Just a debug message , relevant for logging'

- command: <the command to execute>

rescue:

- debug: msg = 'There was an exception.. '

- command: <Rescue mechanism for the above exception occurred)

always:

- debug: msg = "this will execute in all scenarios. Always will get logged"

Following is the syntax for exception handling.

* **rescue** and **always** are the keywords specific to exception handling.
* Block is where the code is written (anything to be executed on the Unix machine).
* If the command written inside the block feature fails, then the execution reaches rescue block and it gets executed. In case there is no error in the command under block feature, then rescue will not be executed.
* **Always** gets executed in all cases.
* So if we compare the same with java, then it is similar to try, catch and finally block.
* Here, **Block** is similar to **try block** where you write the code to be executed and **rescue** is similar to **catch block** and **always** is similar to **finally**.

## Loops

Below is the example to demonstrate the usage of Loops in Ansible.

The tasks is to copy the set of all the war files from one directory to tomcat webapps folder.

Most of the commands used in the example below are already covered before. Here, we will concentrate on the usage of loops.

Initially in the 'shell' command we have done ls \*.war. So, it will list all the war files in the directory.

Output of that command is taken in a variable named output.

To loop, the 'with\_items' syntax is being used.

with\_items: "{{output.stdout\_lines}}" --> output.stdout\_lines gives us the line by line output and then we loop on the output with the with\_items command of Ansible.

Attaching the example output just to make one understand how we used the stdout\_lines in the with\_items command.

---

#Tsting

- hosts: tomcat-node

tasks:

- name: Install Apache

shell: "ls \*.war"

register: output

args:

chdir: /opt/ansible/tomcat/demo/webapps

- file:

src: '/opt/ansible/tomcat/demo/webapps/{{ item }}'

dest: '/users/demo/ajay/{{ item }}'

state: link

with\_items: "{{output.stdout\_lines}}"

Loop

## Blocks

The playbook in totality is broken into blocks. The smallest piece of steps to execute is written in block. Writing the specific instruction in blocks helps to segregate functionality and handle it with exception handling if needed. Example of blocks is covered in variable usage, exception handling and loops above.

## Conditionals

Conditionals are used where one needs to run a specific step based on a condition.

---

#Tsting

- hosts: all

vars:

test1: "Hello Ajay"

tasks:

- name: Testing Ansible variable

debug:

msg: "Equals"

when: test1 == "Hello Ajay"

In this case, Equals will be printed as the test1 variable is equal as mentioned in the when condition. “**when”** can be used with a logical OR and logical AND condition as in all the programming languages.



Just change the value of test1 variable from Hello Ajay to say Hello World and see the output.



# Ansible - Advanced Execution

## How to Limit Execution by Tasks

This is a very important execution strategy where one needs to execute only one execution and not the entire playbook. **For example**, suppose you only want to stop a server (in case a production issue comes) and then post applying a patch you would like to only start the server.

Here in original playbook stop and start were a part of different roles in the same playbook but this can be handled with the usage of tags. We can provide different tags to different roles (which in turn will have tasks) and hence based on the tags provided by the executor only that specified role/task gets executed. So for the above example provided, we can add tags like the following −

- {role: start-tomcat, tags: ['install']}}

The following command helps in using tags −

ansible-playbook -i hosts <your yaml> --tags "install" -vvv

With the above command, only the start-tomcat role will be called. The tag provided is case-sensitive. Ensure exact match is being passed to the command.

## How to Limit Execution by Hosts

There are two ways to achieve the execution of specific steps on specific hosts. For a specific role, one defines the hosts - as to which specific hosts that specific role should be run.

**Example**

- hosts: <A>

environment: "{{your env}}"

pre\_tasks:

- debug: msg = "Started deployment.

Current time is {{ansible\_date\_time.date}} {{ansible\_date\_time.time}} "

roles:

- {role: <your role>, tags: ['<respective tag>']}

post\_tasks:

- debug: msg = "Completed deployment.

Current time is {{ansible\_date\_time.date}} {{ansible\_date\_time.time}}"

- hosts: <B>

pre\_tasks:

- debug: msg = "started....

Current time is {{ansible\_date\_time.date}} {{ansible\_date\_time.time}} "

roles:

- {role: <your role>, tags: ['<respective tag>']}

post\_tasks:

- debug: msg = "Completed the task..

Current time is {{ansible\_date\_time.date}} {{ansible\_date\_time.time}}"

As per the above example, depending on the hosts provided, the respective roles will only be called. Now my hosts A and B are defined in the hosts (inventory file).

### Alternate Solution

A different solution might be defining the playbook's hosts using a variable, then passing in a specific host address via **--extra-vars** −

# file: user.yml (playbook)

---

- hosts: '{{ target }}'

user: ...

playbook contd….

### Running the Playbook

ansible-playbook user.yml --extra-vars "target = "<your host variable>"

If {{ target }} isn't defined, the playbook does nothing. A group from the hosts file can also be passed through if need be. This does not harm if the extra vars is not provided.

### Playbook targeting a single host

$ ansible-playbook user.yml --extra-vars "target = <your hosts variable>" --listhosts

# Ansible Troubleshooting

## Debug and Register

These two are the modules available in Ansible. For debugging purpose, we need to use the two modules judiciously. Examples are demonstrated below.

## Use Verbosity

With the Ansible command, one can provide the verbosity level. You can run the commands with verbosity level one (-v) or two (-vv).

## Important Points

In this section, we will go through a few examples to understand a few concepts.

If you are not quoting an argument that starts with a variable. For example,

vars:

age\_path: {{vivek.name}}/demo/

{{vivek.name}}

This will throw an error.

### Solution

vars:

age\_path: "{{vivek.name}}/demo/" – marked in yellow is the fix.

How to use register -> Copy this code into a yml file say test.yml and run it

---

#Tsting

- hosts: tomcat-node

tasks:

- shell: /usr/bin/uptime

register: myvar

- name: Just debugging usage

debug: var = myvar

When I run this code via the command Ansible-playbook -i hosts test.yml, I get the output as shown below.

If you see the yaml , we have registered the output of a command into a variable – **myvar** and just printed the output.

The text marked yellow, tells us about property of the variable –myvar that can be used for further flow control. This way we can find out about the properties that are exposed of a particular variable. The following debug command helps in this.

$ ansible-playbook -i hosts test.yml

PLAY [tomcat-node] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Monday 05 February 2018 17:33:14 +0530 (0:00:00.051) 0:00:00.051 \*\*\*\*\*\*\*

ok: [server1]

TASK [command] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Monday 05 February 2018 17:33:16 +0530 (0:00:01.697) 0:00:01.748 \*\*\*\*\*\*\*

changed: [server1]

TASK [Just debugging usage] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Monday 05 February 2018 17:33:16 +0530 (0:00:00.226) 0:00:01.974 \*\*\*\*\*\*\*

ok: [server1] => {

"myvar": {

"changed": true,

"cmd": "/usr/bin/uptime",

"delta": "0:00:00.011306",

"end": "2018-02-05 17:33:16.424647",

"rc": 0,

"start": "2018-02-05 17:33:16.413341",

"stderr": "",

"stderr\_lines": [],

"stdout": " 17:33:16 up 7 days, 35 min, 1 user, load average: 0.18, 0.15, 0.14",

"stdout\_lines": [

" 17:33:16 up 7 days, 35 min, 1 user, load average: 0.18, 0.15, 0.14"

]

}

}

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

server1 : ok = 3 changed = 1 unreachable = 0 failed = 0

## Common Playbook Issues

In this section, we will learn about the a few common playbook issues. The issues are −

* Quoting
* Indentation

Playbook is written in yaml format and the above two are the most common issues in yaml/playbook.

Yaml does not support tab based indentation and supports space based indentation, so one needs to be careful about the same.

**Note** − once you are done with writing the yaml , open this site (<https://editor.swagger.io/>) and copy paste your yaml on the left hand side to ensure that the yaml compiles properly. This is just a tip.

# Hands-on Lab

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as inventory. For our first challenge, we are going to create an inventory to be used in the following challenges.

The inventory tells what nodes are out there to be used by Ansible, what credentials need to be used to connect to them, how the nodes are grouped, and other necessary variables.

An example inventory:

[all:vars]

ansible\_user=rhel

ansible\_password=SomethingSomething

ansible\_port=22

[web]

websrv1

websrv2

[db]

db\_1 ansible\_host=11.22.33.44

db\_2 ansible\_host=44.55.66.77

Playbooks are files which describe the desired configurations or steps to implement on managed hosts. Playbooks can change lengthy, complex administrative tasks into easily repeatable routines with predictable and successful outcomes.

A playbook can have multiple plays and a play can have one or multiple tasks. In a task a module is called, like the modules in the previous chapter. The goal of a play is to map a group of hosts. The goal of a task is to implement modules against those hosts.

Playbooks are text files written in YAML format and therefore need:

* to start with three dashes (---)
* proper indentation using spaces and not tabs!

There are some important concepts:

* hosts: the managed hosts to perform the tasks on
* tasks: the operations to be performed by invoking Ansible modules and passing them the necessary options.
* become: privilege escalation in Playbooks, same as using -b in the ad hoc command.

Good Playbooks are idempotent, so if a Playbook is run once to put the hosts in the correct state, it should be safe to run it a second time and it should make no further changes to the hosts.

In this challenge you create a playbook to set up an Apache web server in three steps:

1. Install httpd package
2. Enable/start httpd service
3. Copy over an web.html file to each web host

The Playbook makes sure the package containing the Apache web server is installed.

There is a best practice on the preferred directory structures for playbooks. We strongly encourage you to read and understand these practices as you develop your Ansible ninja skills. That said, our playbook today is very basic and creating a complex structure will just confuse things.

Instead, we are going to use the simple directory structure for our playbook and add just a couple of files to it.

## Create an inventory

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as inventory. For our first challenge, we are going to create an inventory to be used in the following challenges.

**Steps**:

On your control host, we are using a simple directory structure in our home directory for the inventory and future playbooks, the directory ansible-files.

Open the editor tab, open the directory ansible-files and in that directory create the file hosts with the content:

[web]

node1

node2

## Creating a Directory Structure and your Playbook

Next, we will write the playbook. In the editor, create a file called apache.yml in the directory ansible-files with the following content.

---

- name: Apache server installed

hosts: node1

become: yes

This shows one of Ansible’s strengths: The Playbook syntax is easy to read and understand. In this Playbook:

* A name is given for the play via name:
* The host to run the playbook against is defined via hosts:
* We enable user privilege escalation with become:

Now that we’ve defined the play, let’s add a task to get something done. We will add a task in which yum will ensure that the Apache package is installed in the latest version. Modify the file so that it looks like the following listing:

---

- name: Apache server installed

hosts: node1

become: yes

tasks:

- name: latest Apache version installed

yum:

name: httpd

state: latest

In the added lines:

* We started the tasks part with the keyword tasks:
* A task is named and the module for the task is referenced. Here it uses the yum module.
* Parameters for the module are added:
  + name: to identify the package name
  + state: to define the wanted state of the package

## Run it

In the last challenge we created the playbook. Now it is time to launch it!

We do this with the command ansible-playbook - one of the central commands of Ansible on the command line!

During the task you might be wondering how ansible-playbook actually knows where to find the hosts file - this is configured in the file /home/rhel/.ansible.cfg which was pre-created in this environment. If you have time have a look at it, it should be fairly self-explanatory.

**Steps**:

Ansible Playbooks are executed using the ansible-playbook command on the control node. We also must tell Ansible where to find the inventory. Before you run a new Playbook it’s a good idea to check for syntax errors. On the control host, change into the ansible-files directory and execute the syntax check:

cd ansible-files

ansible-playbook --syntax-check apache.yml

Now you should be ready to run your playbook:

ansible-playbook apache.yml

You might have to accept the SSH fingerprint during this first connection to the host.

The output should not report any errors but provide an overview of the tasks executed and a play recap summarizing what has been done. There is also a task called “Gathering Facts” listed there: this is an built-in task that runs automatically at the beginning of each play. It collects information about the managed nodes. Exercises later on will cover this in more detail.

Connect to node1 via SSH to make sure Apache has been installed:

ssh node1

Use the command rpm -qi httpd to verify httpd is installed:

rpm -qi httpd

The output lists the name, version, and other details about the package.

Run the Playbook a second time and compare the output: The output changed from “changed” to “ok”, and the color changed from yellow to green. Also, the “PLAY RECAP” is different now. This makes it easy to spot what Ansible actually did.

## Extend the Playbook

By default, Ansible executes each task in order, one at a time, against all machines matched by the host pattern. Each task executes a module with specific arguments. When a task has executed on all target machines, Ansible moves on to the next task.

When you run a playbook, Ansible returns information about connections, the name lines of all your plays and tasks, whether each task has succeeded or failed on each machine, and whether each task has made a change on each machine. At the bottom of the playbook execution, Ansible provides a summary of the nodes that were targeted and how they performed. General failures and fatal “unreachable” communication attempts are kept separate in the counts.

**Steps**:

To add another task to the playbook, open the editor and open the file apache.yml again. Add the following task at the end of the file:

- name: Apache enabled and running

service:

name: httpd

enabled: true

Make sure that the task is indented the same way as the previous task!

Switch to the control tab, and launch the playbook again in the corresponding directory ansible-files:

ansible-playbook apache.yml

Note how the first task just reports OK, while the second reports the status changed.

## Extend the playbook more: copy files

Besides pure interactions on the target node there is also the possibility to copy data between the controlling instance and the managed nodes. The copy module can be used to transfer data.

**Steps**:

In the editor, create a file directory files underneath the directory ansible-files. Inside the files directory, create a html file called web.html with the following content

<body>

<h1>Apache is running fine, thanks to Ansible!</h1>

</body>

Now go back to the file apache.yml and add a task to copy this file to the managed node:

- name: copy web.html

copy:

src: web.html

dest: /var/www/html/index.html

Note here that the source does not mention the directory files! The files directory is a default place that is searched for a given source automatically.

On the control host, change to the directory ansible-files execute the playbook again:

cd ansible-files

ansible-playbook apache.yml

As you see, the file is deployed.

## From 1 to Many

The real power of Ansible is to apply the same set of tasks reliably to many hosts.

So, what about changing the apache.yml Playbook to run on node2 and node3?

To do that, the playbook will not target an individual node in the future, but instead a group of nodes. Ansible automatically identifies all hosts in the group via the inventory.

As a best practice Ansible in real life applications playbooks are usually written addressing the group all - the limitation to certain nodes or groups is enforced at execution time via command line flags or via tools like Ansible controller.

**Steps**:

In the editor, open the apache.yml file and Change the hosts line value from node1 to web. That way during the next run of the playbook, instead of one host, Ansible targets an entire group.

Run the playbook afterwards on the control host with:

cd ansible-files

ansible-playbook apache.yml

Once the execution was successful, ssh to node2 and verify that the service is indeed running:

systemctl status httpd

## Adding Variables to the Mix

Ansible supports variables to store values that can be used in Playbooks. Variables can be defined in a variety of places and have a clear precedence. Ansible substitutes the variable with its value when a task is executed.

Variables are referenced in Ansible Playbooks by placing the variable name in double curly braces:

Here comes a variable {{ variable1 }}

Variables and their values can be defined in various places: the inventory, additional files, on the command line, etc.

The recommended practice to provide variables in the inventory is to define them in files located in two directories named host\_vars and group\_vars:

* To define variables for a group “servers”, a YAML file named group\_vars/servers.yml with the variable definitions is created.
* To define variables specifically for a host node1, the file host\_vars/node1.yml with the variable definitions is created.

Host variables take precedence over group variables (more about precedence can be found in the [docs](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)).

**Steps**:

We want to save variables for certain hosts, and for the full group.

In the editor, in the directory ansible-files, create two directories, group\_vars and host\_vars:

We are going to set values for the variable stage. Create the file ~/ansible-files/group\_vars/web.yml with this content:

---

stage: dev

Create the file ~/ansible-files/host\_vars/node1.yml with this content:

---

stage: prod

Based on the stage, the playbook should copy a different file. Let's create two stage files. First the file ~/ansible-files/files/prod\_web.html with the content:

<body>

<h1>This is a production webserver, take care!</h1>

</body>

Second create the file ~/ansible-files/files/dev\_web.html with the content:

<body>

<h1>This is a development webserver, have fun!</h1>

</body>

Modify the copy task of the existing apache.yml playbook to this:

- name: copy web.html

copy:

src: "{{ stage }}\_web.html"

dest: /var/www/html/index.html

**Note**: You could copy the html files into /usr/share/nginx/html which is the default location for the web server’s index.html to overwrite it and then try the curl command (*given below*).

On the control host, run the playbook:

cd ansible-files

ansible-playbook apache.yml

Test and compare the results:

curl http://node1

curl http://node2

## Decide what to do - using conditionals

In a playbook, you may want to execute different tasks, or have different goals, depending on the value of a fact (data about the remote system), a variable, or the result of a previous task. You may want the value of some variables to depend on the value of other variables. Or you may want to create additional groups of hosts based on whether the hosts match other criteria. You can do all of these things with conditionals.

The simplest conditional statement applies to a single task. Create the task, then add a when statement that applies a test: when: ansible\_selinux.status == "enabled".

**Steps**:

Imagine we are extending our setup: the web servers will get an MTA to be able to send status mails, and we add a backend database.

To do that, open the hosts file in the ansible-files directory in your editor. Change the file to add another entry:

[web]

node1

node2

[database]

node3

Next, we will write a playbook installing the services depending on what group the node is in. In the directory ansible-files, create the file extended\_services.yml. First, we add the task to install the MTA Postfix on all node of the group web:

---

- name: Install extended services

hosts: all

become: true

tasks:

- name: Install MTA in web group

yum:

name: postfix

state: latest

when: inventory\_hostname in groups["web"]

Next, add a task installing the database on the node of the group database:

- name: Install database in corresponding group

yum:

name: postgresql

state: latest

when: inventory\_hostname in groups["database"]

On the control host, execute the playbook:

cd ansible-files

ansible-playbook extended\_services.yml

## Using Loops

Loops enable us to repeat the same task over and over again. For example, lets say you want to create multiple users. By using an Ansible loop, you can do that in a single task.

Loops can also iterate over more than just basic lists. For example, if you have a list of users with their corresponding group, loop can iterate over them as well.

Find out more about loops in the [Ansible Loops documentation](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html).

**Steps**:

To show the loops feature we will generate three new users on node2. For that, open the editor and create the file loop\_users.yml. We will use the user module to generate the user accounts.

---

- name: Ensure users

hosts: node2

become: true

tasks:

- name: Ensure three users are present

user:

name: "{{ item }}"

state: present

loop:

- dev\_user

- qa\_user

- prod\_user

On the control host, run the playbook:

cd ansible-files

ansible-playbook loop\_users.yml

Understand the playbook and the output:

* The names are not provided to the user module directly. Instead, there is only a variable called {{ item }} for the parameter name.
* The loop keyword lists the actual usernames. Those replace the {{ item }} during the actual execution of the playbook.
* During execution the task is only listed once, but there are three changes listed underneath it.

## Templates

Ansible uses [Jinja2](http://jinja.pocoo.org/)templating to modify files before they are distributed to managed hosts. Jinja2 is one of the most used template engines for Python.

When a template for a file has been created, it can be deployed to the managed hosts using the template module, which supports the transfer of a local file from the control node to the managed hosts.

A typical ending for a file to indicate that it is a template file is .j2. Though this is strictly speaking not necessary, it is established practice.

**Steps**:

First, in the editor, create the directory templates to hold template resources underneath ~/ansible-files/.

In there, create the file motd-facts.j2:

Welcome to {{ ansible\_hostname }}.

{{ ansible\_distribution }} {{ ansible\_distribution\_version}}

deployed on {{ ansible\_architecture }} architecture.

The template file contains the basic text that will later be copied over. It also contains variables which will be replaced on the target machines individually.

Next, we need a playbook to use this template. In the ~/ansible-files/ directory create the Playbook motd-facts.yml:

---

- name: Fill motd file with host data

hosts: node2

become: true

tasks:

- template:

src: motd-facts.j2

dest: /etc/motd

owner: root

group: root

mode: 0644

Execute it on the control host:

cd ansible-files

ansible-playbook motd-facts.yml

Login to node2 via SSH and check the message of the day content. It will be displayed as soon as you successfully ssh to the node.

# References

<https://www.guru99.com/ansible-tutorial.html#1>

<https://www.tutorialspoint.com/ansible/index.htm>

<https://phoenixnap.com/kb/install-ansible-on-windows>

<https://docs.ansible.com/ansible/latest/user_guide/quickstart.html>

<https://www.ansible.com/products/ansible-training?hsLang=en-us>

Videos:

<https://www.youtube.com/watch?v=EcnqJbxBcM0> (Very good)