**ed Hat, Centos, or Linux Basic administration tasks (1 question)**

100%

**Ansible Configurations (1 question)**

100%

**Ansible AdHoc (1 question)**

100%

**Ansible Roles (1 question)**

100%

**Download roles based on requirements files (1 question)**

100%

**Ansible Vault (1 question)**

100%

**Ansible Conditions (4 questions)**

100%

**Jinja2 Templates (1 question)**

100%

**Ansible facts (1 question)**

100%

**System Roles (1 question)**

100%

**selinux modules (1 question)**

[**DEVOPS**](https://www.linuxteacher.com/category/devops/)

**Let’s learn ansible and pass RHCE EX294 Exam Part 01**

**I’m creating this ansible tutorial article series to focus to the RedHat certified engineer new**[**exam EX294**](https://www.redhat.com/en/services/training/ex294-red-hat-certified-engineer-rhce-exam-red-hat-enterprise-linux-8)**.**

**This exam is mainly focused on the ansible.**

**so I,m starting from very basics and continue until cover the all the details which need to pass the exam. so this is the first article of the series. so let’s start.**

**what is ansible?**

**Ansible is an automation tool.**

**just think like this.**

**you are a system engineer and you want to install Nginx in 20 ubuntu servers. so if you do it manually you have to do it like 1 or 2 hours.**

**But if you use a tool like an ansible you can do it like within 5 minis by just using one command. Just take the idea we will discuss this later with details.**

**Please check the below diagram.**

* **A picture containing diagram, line, text, screenshot

  Description automatically generated**

**The Ansible software is only needed to install in one server.**

**in the above image, you can see the Ansible controller node, which is the server we install ansible.**

**then that ansible controller node can access the other servers through SSH. no need to install any agent in other servers. it is a huge benefit of ansible.**

**according to our Nginx example, we are running ansible command in “ansible control node” and Nginx will be installed in all 3 “managed hosts”**

**ok, lets setup ansible.**

**Setup ansible on Centos 8**

**we are going to create one ansible Control Node and three managed hosts like in the above image.**

**to create an ansible control node we have to install epel-release first. please note no need to install this package in other servers.**

**Run this command to install epel-release on centos 8**

****

|  |  |
| --- | --- |
| **1** | **yum install epel-release** |

****

|  |  |
| --- | --- |
| **1** | **yum install ansible** |

|  |  |
| --- | --- |
| **1** | **ansible --version** |

**if the installation is successful, you will get an output like this**

****

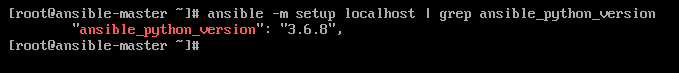
**ansible is based on python. so all the servers including ansible master and managed nodes must have python installed.**

**But you no need to worry about it. Python is installed by default in RHEL 8 servers.**

**you can verify the python version on the ansible master node by executing the below command.**

****

|  |  |
| --- | --- |
| **1** | **ansible -m setup localhost | grep ansible\_python\_version** |

****

**ok cool. you have learned some basics. we will cover all about**[**ansible inventory in the next tutorial**](https://www.linuxteacher.com/buiding-an-ansible-inventory-rhce-ex294-exam-part-02/)**.**

**Buiding an Ansible inventory RHCE EX294 Exam Part 02**

In the [previous article](https://www.linuxteacher.com/lets-learn-ansible-and-pass-rhce-ex294-exam-part-01/), we made an introduction to Ansible and walked you through its installation. In this second part series, we will focus on creating an Ansible inventory and defining hosts inside the inventory. At the end of this chapter, you should be able to comprehend the various concepts about Ansible inventory concepts as well as manage a static inventory.

**What’s an Ansible inventory?**

A good place to start would be to define what an ansible inventory is. In simple terms. **An Ansible inventory is a database of hosts or target systems that will be managed by Ansible**. It’s in the inventory that you define all the servers or network devices that you are going to configure.

Hosts can be defined individually or grouped into groups which are then managed collectively. A host can be a member of several groups. Groups can also have child groups which we shall discuss further as we go along. Additionally, you can specify variables which can be used to define hosts and groups.

There are 2 types of inventories:

**Static Inventory:** This is defined in a text file in INI format, which usually resides on the Ansible controller. By default, the default path for inventory files in the /etc/ansible/hosts directory.

**Dynamic inventory:** This is usually a script-generated file using external cloud providers such as AWS and Azure. A dynamic inventory pulls information from multiple sources.

**Defining hosts in a static Ansible inventory**

As stated earlier, a static inventory is a text file which usually takes the INI format. It can be edited using conventional text editors such as vim, and nano.

In its basic form, the file the INI style explicitly lists the IP addresses or hostnames as shown:



|  |  |
| --- | --- |
| 1  2  3  4 | web-server1.company.com  web-server2.company.com  173.82.115.6  173.82.82.100 |

In a large network of multiple hosts, hosts can be grouped into **host groups**. This provides a much better way of managing them.

**NOTE:**

By default, 2 host groups exist:

* The ‘**all**‘ host group which comprises of hosts which are explicitly listed inside the inventory
* The **ungrouped** host group. This comprises of all hosts specified in the inventory that are not members of any host group.

Picture a scenario where you have 4 web servers, 2 database servers and 2 load balancers. Instead of listing them individually in the inventory file, you can group them using group names enclosed in square brackets [].

Using our conceptual setup, we are going to organize our hosts into webservers, db-servers. and load-balancers host groups.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <strong>[web-servers]</strong>  173.82.202.201  173.80.187.242  173.82.150.145  173.82.115.6    <strong>[db-servers]</strong>  db01.company1.com  db02.company2.com    <strong>[load-balancers]</strong>  lb-01.company.com  lb-02.company.com |

Additionally, a host can belong to multiple host groups depending on several factors such as the physical location, role of the host and so on.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <strong>[web-servers]</strong>  173.82.202.201  173.80.187.242  173.82.150.145  173.82.115.6    <strong>[db-servers]</strong>  db01.company1.com  db02.company2.com    <strong>[ohio-datacenter]</strong>  173.82.202.201  173.80.187.242  db01.company1.com    <strong>[miami-datacenter]</strong>  173.82.150.145  173.82.115.6  db02.company2.com    <strong>[load-balancers]</strong>  lb-01.company.com  lb-02.company.com |

**Nested host groups**

Additionally, you can have host groups that belong to other host groups. This is made possible by having a host group name with the suffix :childrenas shown.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <strong>[ohio-datacenter]</strong>  173.82.202.201  173.80.187.242  db01.company1.com    <strong>[miami-datacenter]</strong>  173.82.150.145  173.82.115.6  db02.company2.com    <strong>[usa:children]</strong>  ohio-datacenter  miami-datacenter |

From the above example, the host group **USA** has 2 child groups which are **ohio-datacenter**and **miami-datacenter**.

If you are in doubt about the hosts in your inventory use the syntax below:

**Verifying an Ansible inventory**

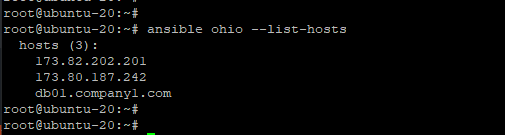


|  |  |
| --- | --- |
| 1 | $ ansible  host-group  --list-hosts |

For example, in my inventory, to see the hosts listed under the host group **ohio,** run the command:



|  |  |
| --- | --- |
| 1 | $ ansible ohio --list-hosts |



To list managed hosts that are not part of a host group, use the **ungrouped** option as follows:



|  |  |
| --- | --- |
| 1 | $ ansible <strong>ungrouped</strong> --list-hosts |

A black screen with white text

Description automatically generated with low confidence

If you specify a host group that does not have any managed hosts specified, you will get an alert informing you that there are no hosts under the particular host group.

A picture containing screenshot, text, font

Description automatically generated

**Specifying a custom inventory file**

By default, the inventory file is the /etc/ansible/hosts file. However, you can create your own inventory file and specify its location using the -i pathname or --inventory pathname option. This is important when you want to specify a different inventory file during playbook execution:

For example, to display managed hosts listed under a host group in an inventory file sitting in the current working directory, use the syntax.

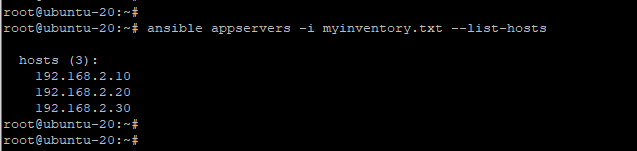


|  |  |
| --- | --- |
| 1 | $ ansible hostgroup -i inventory\_file --list-hosts |

The example below lists all the managed hosts under the **appservers** host group using the **myinventory.txt** inventory file located in the current working directory.



|  |  |
| --- | --- |
| 1 | $ ansible appservers -i mynventory.txt --list-hosts |



The above command overrides the default inventory file located at /etc/ansible/hosts file and the playbook file will now reference the specified inventory file.

**Managing Ansible Configuration files**

A user can determine how Ansible runs by customizing the settings of the Ansible configuration file which is located on the control node. In Ansible, the default configuration file is ansible.cfg file located at /etc/ansible/ansible.cfg.

Additionally, ansible can pick the ansible.cfg configuration file from 3 other possible locations.

**1) Using the ~/.ansible.cfg file**

If the Ansible configuration file exists in the user’s home directory ~/.ansible.cfg , then this file overrides the default configuration file in the /etc/ansible/ansible.cfg path.

**2) Using the ./ansible.cfg file in the current directory when running Ansible**

If the ansible.cfg file is present in the working directory where Ansible command is being invoked, it is given first priority instead of the user’s personal file or the global file /etc/ansible/ansible.cfg.

**3) Using the ANSIBLE\_CONFIG environment variable**

As earlier mentioned, The ansible.cfg configuration file can be placed in different directories to suit your needs. However, this avenue is quite restrictive and it may prove a challenge managing the configuration files. A better and viable option is to define the configuration file’s location in the **ANSIBLE\_CONFIG** environment variable. When the variable is defined, this takes precedence over the previously discussed configuration files.

**CONFIGURATION FILE PRECEDENCE**

Due to the myriad of directories in which the Ansible configuration file can be placed, it can be somewhat confusing which configuration file takes precedence. To have an accurate picture of which configuration file is currently in use, invoke the **ansible --version** command as shown.



|  |  |
| --- | --- |
| 1 | $ ansible --version |

A computer screen with white text

Description automatically generated with low confidence

The first line of the output gives us information about the version of Ansible – In this case Ansible version 2.9.6. The second line specifies the default path of the configuration file which is **/etc/ansible/ansible.cfg**

The **ANSIBLE\_CONFIG** variable always takes the highest precedence. If this environment variable exists on your current shell. it overrides all the other configuration files. So why would you want to use an environment variable? This comes in handy when you have multiple projects and you want them to use a global or specific file other than the one present in the /etc/ansible directory.

Second in line is placing the ansible.cfg file in your current working directory.

If Ansible doesn’t locate the file in your current working directory, it then proceeds to check the file in the user’s home directory – ~/.ansible.cfg.

If there isn’t one there either, It finally grabs the /etc/ansible/ansible.cfg file.

In summary, here’s the precedence:

1. The ANSIBLE\_CONFIG environment variable.
2. The **ansible.cfg** located in your current working directory.
3. **.ansible.cfg** file in the home directory).
4. **/etc/ansible/ansible.cfg** file

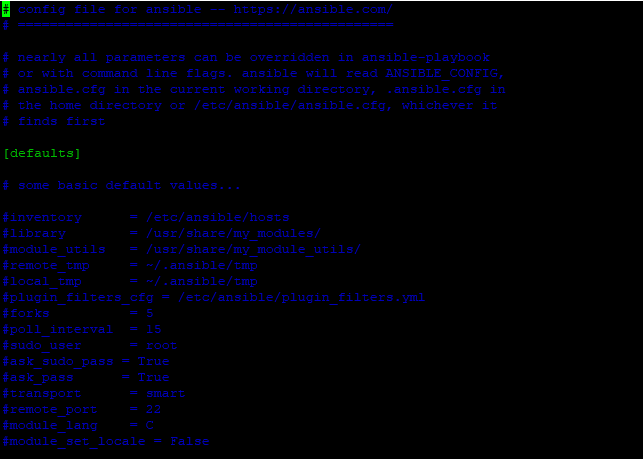
**EXPLORING SETTINGS IN THE ANSIBLE CONFIGURATION FILE**

The Ansible configuration file is the heartbeat of Ansible. It determines how Ansible behaves during playbook execution. With that in mind, we are going to explore the salient parameters. Let’s dive in.

The Ansible configuration file consists of 10 distinct sections. However, for basic operations and the for scope of this guide, we will focus on 2 main sections:

The **[defaults]** section – Contains the default settings for Ansible.

The **[privilege\_escalation]** – Determines how privilege escalation is performed.



Let’s concentrate on the following key attributes:

**inventory** – As the name suggests, this defines the path to the inventory file.

**remote\_user** – This is the name of the login user on the managed host. If unspecified, the current user’s name is used instead.

**ask\_pass** – Determines whether or not to prompt for a user password using SSH. This directive is ignored in the case of SSH public key authentication.

**become** – Specifies whether to switch to a user on the managed host (In most cases the root user) after establishing an SSH connection.

**become\_method** – Specifies the user to switch on to when logged in to the managed host ( Usually this is the sudo user. You can also pass the su option.

**become\_user** – Defines the user to ‘become’ or switch to once logged in.

**become\_ask\_pass** – Determines whether to prompt for a password when switching users using the become\_method option.

**MANAGING CONNECTIONS**

One of the key reasons for editing the configuration file is to determine which users and methods Ansible will use to control and configure managed hosts. Key information required by Ansible includes:

The inventory file that lists all the managed hosts, grouped or otherwise.

The connection protocols to connect to the managed hosts. By default, this is usually SSH. Also, it comes in handy if any non-standard port is to be defined.

The remote user to use on the remote hosts. This could either be root or any other unprivileged user.

If you intend to use the unprivileged user on the remote host, Ansible needs to know if to escalate privileges and how to go about it (For example, by invoking sudo )

Whether or prompt to go ahead and prompt for an SSH password or use a sudo password to acquire privileges.

**Inventory location**

Within the **[defaults]** section the path to the inventory file is the first to be specified and it’s the default location is **/etc/ansible/hosts**. It’s usually commented, but if you need to specify the path to your static inventory file, uncomment and provide the inventory path.

For example, the line below places the inventory file to the home directory.



|  |  |
| --- | --- |
| 1 | <strong>inventory  =  ~/</strong>inventory.ini |

**Connection settings**

By default, Ansible uses the SSH protocol in Ansible, thanks to its strong encryption which keeps information private and confidential. The parameters that determine how you connect via SSH are in the **[defaults]** section.

Usually, Ansible tries to connect to the remote managed host using the username of the local user who is invoking the Ansible command. However, if you need to specify another remote user, be sure to set the **remote\_user** parameter to the user on the managed host.

If SSH public key authentication has been configured by the local user, then Ansible will automatically log in without an issue. In that case, you can set **ask\_pass=false** To generate SSH keys, simply run the command:



|  |  |
| --- | --- |
| 1 | $ ssh-keygen |

Press ENTER all the way to save the SSH public and private key to your home directory at **~/.ssh**

A screenshot of a computer

Description automatically generated

To copy the public key to the managed host, use the **ssh-copy-id** command as shown:



|  |  |
| --- | --- |
| 1 | $ ssh-copy-id root@remote-host-ip |

A screenshot of a computer program

Description automatically generated with low confidence

Thereafter, you can log in to the managed host without providing a password. In the example below, we are logging to a Ubuntu managed host.



|  |  |
| --- | --- |
| 1 | $ ssh 173.82.104.69 |

A computer screen with white text

Description automatically generated with low confidence

**Escalating privileges**

If you are connecting to Ansible as an underprivileged user, you will need elevated privileges to carry out administrative tasks on the managed host. This can be configured in the **[privilege\_escalation]** section.

To enable escalation by default, set the parameter **become=true** However, note that this can be overridden when invoking Ansible ad hoc commands or executing playbooks.

The **become\_method** parameter defines how the escalation will be done. The default value for this is usually sudo. Similarly, the **become\_user** directive dictates which user to escalate to, and by default, this is the root user.

If a password is required to escalate privileges, then set the **become\_ask\_pass=true** directive.

**Non-ssh connections**

If localhost is not specified in your inventory, Ansible, by default, creates an implicit localhost entry to allow users to invoke ad hoc commands and run playbooks targeting the localhost system. The localhost entry is usually not present in the inventory. However, Ansible can still connect to your localhost system using a unique **local**connection.

You can list the localhost entry as shown.



|  |  |
| --- | --- |
| 1 | $ ansible localhost --list-hosts |

A black background with white text

Description automatically generated with low confidence

The local connection type overrides the **remote\_user** directive and executes commands diretly on your local system. Incase privilege escalation is required, it uses the**sudo**directive from the user that invokes the Ansible command.

**Configuration file comments**

Comments, as you would find in any programming language, give a brief description about what a section of code does. Similarly, in Ansible, you will find comments which start with a hash sign ( **#** ) or semi-colon ( **;** ).

The hash sign comments out the entire line. Take note that it shouldn’t be on the same line with a parameter or directive.

Meanwhile, the semi-colon sign comments out anything on the right on a line. You can have it on the same line as a directive provided that the directive is to the left on the line.

**RUNNING AD-HOC COMMANDS WITH ANSIBLE**

An Ansible ad hoc command is a way of quickly executing simple tasks without writing a playbook file. It’s a simple, one-line command that comes in handy when you want to run a quick test or implement a change. For example, you could use an ad hoc command to check the disk space, uptime, ensure whether a user exists or whether a certain service is running. Those are just a few examples. You can do a whole lot more.

Ad hoc commands are simple but they have their limitations as well. If you want to realize the full potential of Ansible, you would want to leverage the playbook files. However, for simple tasks, ad hoc commands are most recommended.

**Running Ad hoc commands**

The syntax for running Ansible ad hoc commands is as shown.



|  |  |
| --- | --- |
| 1 | $ ansible host-pattern -m module\_name -a "module\_options" |

Let’s break down the command options:

The **host-pattern** option specifies the managed host system on which the ad hoc command will run. This should either be an individual host or host group defined in the inventory file. Earlier on, we saw how you can list the available hosts in the inventory file using the **--list-hosts** option.



|  |  |
| --- | --- |
| 1 | $ ansible all --list-hosts |

We have also discussed using the -i option to specify a different inventory location other than the default Ansible inventory file.

The **-m** option specifies the Ansible module that should be run on the remote hosts. An Ansible module is a small program that’s usually executed to perform a given task. Some modules are quite simple and require no additional information while others need additional options to specify the operation. For modules that are quoted as a string, the **-a** option is used instead.

The most simple and basic of all modules is the ping module. It does not send an ICMP echo request to the managed host but probes to see whether you can execute Python-based modules on remote hosts.



|  |  |
| --- | --- |
| 1 | $ ansible -m ping all |

The above command checks to see if all the managed hosts in the **/etc/ansible/hosts** file can run standard modules.

A picture containing text, screenshot, font

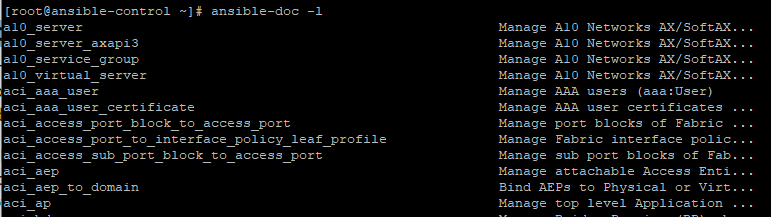
Description automatically generated

**Performing Tasks with Modules Using Ad Hoc Commands**

Ansible packs with hundreds upon hundreds of modules to perform various tasks. To list all the modules on a system, use the ansible command as shown.



|  |  |
| --- | --- |
| 1 | $ ansible-doc -l |



To view detailed documentation of a specific module, use the syntax shown below.

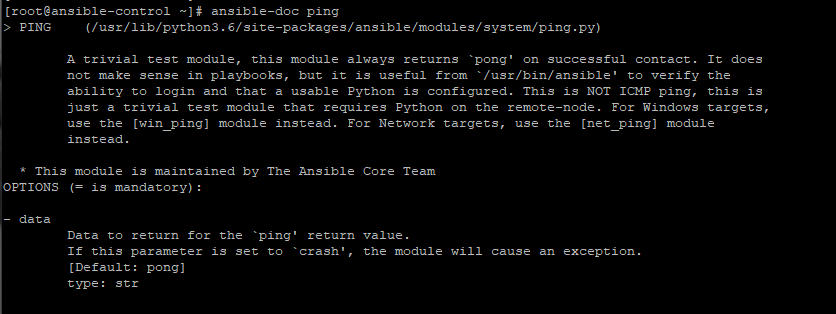


|  |  |
| --- | --- |
| 1 | $ ansible-doc module\_name |

For example, to find in-depth information about the ping module, run the command:



|  |  |
| --- | --- |
| 1 | $ ansible-doc ping |



For more information about Ansible modules, kindly refer to the [Ansible documentation on modules.](https://docs.ansible.com/ansible/2.8/modules/list_of_all_modules.html)

In most situations, you are going to pass arguments to ansible modules using the **-a** flag.

In the example below, we are using the**user** module to create a new user **winnie** on the remote system.



|  |  |
| --- | --- |
| 1 | $ ansible -m user -a 'name=winnie uid=1002 state=present'  173.82.104.69 |

A screen shot of a computer

Description automatically generated with medium confidence

Ansible, modules are **idempotent**. This implies that you can run them multiple times without yielding different results beyond the first operation. For example, in the previous example, if we try again to create the user **winnie**, the command will safely run without any changes since the user already exists.

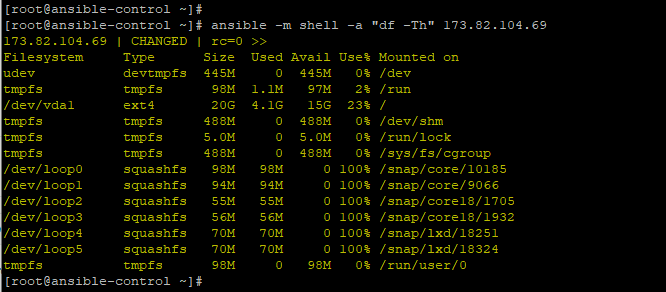
A screen shot of a computer

Description automatically generated with medium confidence

You can also use the **shell** module to run shell commands on the remote managed host. The command below checks the disk space utilization on the managed host.



|  |  |
| --- | --- |
| 1 | $ ansible -m shell -a "df -Th" 173.82.104.69 |



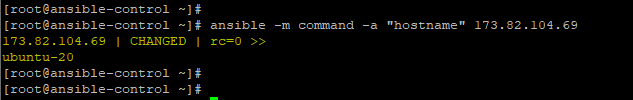
**Running Arbitrary Commands on Managed Hosts**

Apart from the shell module that allows administrators to run commands on managed hosts, another module that allows users to achieve the same result is the**command** module. It is passed as an argument using the **-a** option.

For example, the command below invokes the commands hostname on a managed host system.



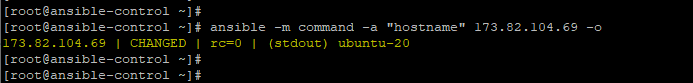
|  |  |
| --- | --- |
| 1 | $ ansible -m command  -a "hostname"  173.82.104.69 |



For improved readability, pass the -o so that the output is displayed on a single line.



|  |  |
| --- | --- |
| 1 | $ ansible -m command  -a "hostname"  173.82.104.69 -o |



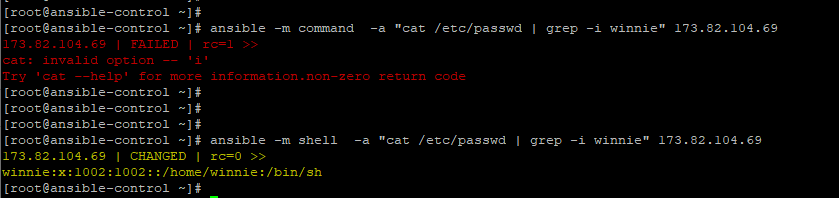
The command module enables system administrators to easily and quickly execute remote commands on remote targets. Take note that the commands are not processed by the shell on remote managed hosts. As such you cannot perform shell operations e.g piping and redirection.

This is where the **shell** module steps in. The shell module allows you to access the shell of managed hosts and perform shell processing tasks such as piping and redirection.

The example below lists the user details of the user **Winnie** in the /etc/passwd file using the **shell** module. Note that the command fails when passing the command module.



|  |  |
| --- | --- |
| 1 | $ ansible -m shell  -a "cat /etc/passwd | grep -i winnie" 173.82.104.69 |



**CONFIGURING CONNECTIONS FOR AD HOC COMMANDS**

The directives for privilege escalation can also be defined for use in ad hoc commands. When these directives are used alongside ad hoc commands, they override the default directives in Ansible’s configuration file.

The table below presents the command-line options for the file directives.

|  |  |
| --- | --- |
| **Configuration file directives** | **Command-line options** |
| **Inventory** | **-i** |
| **remote\_user** | **-u** |
| **become** | **–become, -b** |
| **become\_method** | **–become-method** |
| **become\_user** | **–become-user** |
| **become\_ask\_pass** | —**ask-become-pass, -K** |

**Implement Ansible Playbooks RHCE EX294 Exam Part 03**

In the [previous chapter](https://www.linuxteacher.com/buiding-an-ansible-inventory-rhce-ex294-exam-part-02/), we made an introduction to Ansible and looked at the fundamental topics. These included creating an Ansible inventory, managing configuration files, and running ad hoc commands on managed hosts. In this chapter, we dive deeper and touch base on how to implement Ansible playbooks to automate tasks on manage remote hosts.

At the end of this chapter, you will have a firm grasp of Ansible Playbooks and be able to write your own playbooks and execute them using the **ansible-playbook** command.

**ANSIBLE PLAYBOOKS AND AD HOC COMMANDS**

Sure, you can use ad hoc commands to run commands on a set or multiple remote hosts and achieve the expected results. However, the true power of Ansible lies in learning how to implement Ansible playbooks in order to execute complex tasks on remote targets.

**What is a playbook?**

A Playbook is a text file in YAML format that contains one or more plays to be executed on managed hosts. It is usually saved in a **.yml** format.

A play is an ordered set of tasks which are to be executed on remote hosts defined in the inventory file. A task is an operation to be carried out. A playbook file can be made of one or multiple plays.

**FORMATTING AN ANSIBLE PLAYBOOK FILE**

To get a better understanding of Ansible playbooks, we will preview an ad hoc command from the previous chapter. The ad hoc command creates a new user **winnie** on the remote host.



|  |  |
| --- | --- |
| 1 | $ ansible -m user -a 'name=winnie uid=1002 state:present' 173.82.104.69 |

The ad hoc command can be rewritten in a playbook file as a single task as shown below:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ---    - name: Configure a new user on remote host      hosts:  173.82.104.69        tasks:       - name: Create a new user with UID 1002         user:           name:  winnie           uid:   1002           state: present |

**Structure of a playbook**

From the example above, a playbook file begins with the dashes (**—**). A play comprises**key-value** pairs. Keys within the same play should have a similar indentation. In this playbook example, we have 3 keys: **name**, **hosts,** and **tasks**.

The very first line begins with a dash and space. This indicates the name of the play The name attribute is the first item on the list and takes an arbitrary string as a label. This is basically an identification that specifies what the play does. While optional, the name key comes recommended for clarity purposes.

Next online is the ‘**hosts’**key. As you might have guessed, the ‘**hosts’**key defines the targets against which the playbook will run. The ‘**hosts**‘ attribute can take an IP address(es), domain name, or host groups in the inventory.

Lastly, we have the ‘**tasks**‘ key that specifies the tasks to be executed in order in a play. The task at hand uses the user module to create a new user on the remote host with the name **winnie** and uid **1002**.

The task in this play has 2 keys:

1. N**ame**: This documents the name of the task, and takes a string value. It is recommended to name your tasks for documentation purposes.
2. User: This is the module used to run the tasks. Its arguments are passed as key-value pairs which are children of the module. The arguments, in this case, are name, uid, and **state**.

**Indentations in Playbooks**

As you might have noted, a playbook consists of indentations to structure its data. Although there are no strict rules on the spacing, a few basic rules apply:

1. Elements in the same list must have the same indentation.
2. Children items must be indented more than their parents. Feel free to add more blank lines for better readability.

For indentation, only use the spacebar key on the keyboard and not the TAB key. A play comprises key-value pairs. Keys within the same play should have a similar indentation.

**RUNNING PLAYBOOKS**

Let’s now switch gears and see how you can run Ansible Playbooks. To run playbooks, the ansible-playbook command is used in the syntax shown



|  |  |
| --- | --- |
| 1 | # ansible-playbook  playbook-file.yml |

The example below shows an example of a playbook file – install\_apache.yml which installs the Apache2 web server on a remote host with the IP **173.82.104.69**



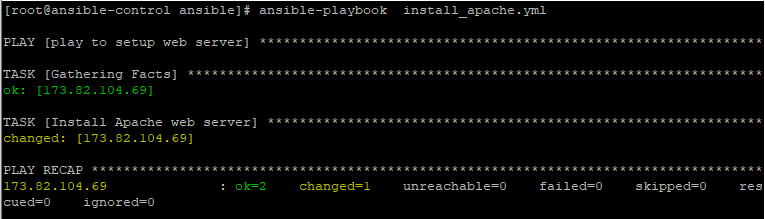
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ---  - name: play to setup web server    hosts: 173.82.104.69      tasks:    - name: Install Apache web server      apt:        name: apache2        state: present |

To execute the playbook, run the command:



|  |  |
| --- | --- |
| 1 | # ansible-playbook install\_apache.yml |

Here’s the output of executing the playbook file.



Let’s briefly discuss the output:

The name of the play is the first to be displayed when the playbook is run. The ‘**Gathering Facts**‘ task is a unique task that is run by the setup module as it gathers information also note that the remote host is indicated as ‘**changed**‘ implying that the operation was a success. This is further evidenced by the ‘PLAY RECAP’ that indicates **ok=2** and **changed=1**

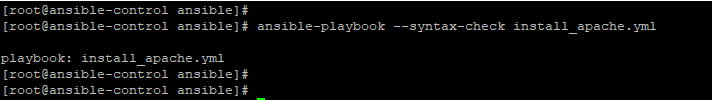
**Syntax verification**

It’s always recommended to check your playbook and ensure that the syntax is correct and that there are no errors. To achieve this, use the **--syntax-check** option as shown.

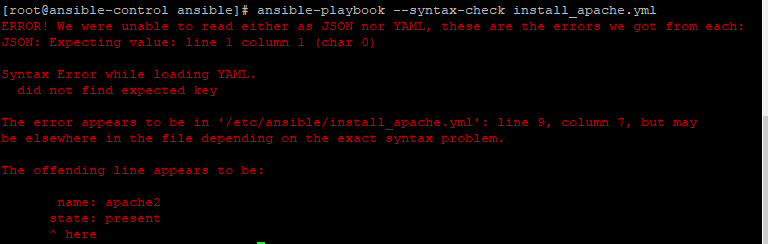


|  |  |
| --- | --- |
| 1 | $ ansible-playbook --syntax-check install\_apache.yml |

If there are no syntax errors, only the playbook file name will be printed as shown.



However, if the syntax check fails, an error will be printed on the terminal as shown.

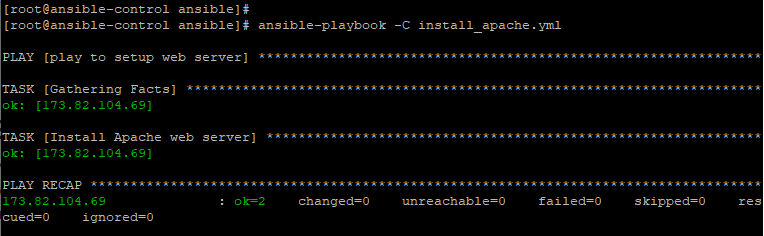


**Perform a dry run**

A dry run simulates the actual playbook execution but does not execute the playbook. It gives you a glance at how the playbook file will run and the changes that will happen during playbook execution. To perform a dry run, use the **-C** option as shown:



|  |  |
| --- | --- |
| 1 | $ ansible-playbook -C install\_apache.yml |



**Increase verbosity**

By default, the ansible-playbook command does not print out detailed task execution when running a playbook file. The **ansible-playbook -v** option provides additional information up to 4 levels during playbook execution.

|  |  |
| --- | --- |
| OPTION | DESCRIPTION |
| **-v** | Displays location of the Ansible configuration file in use. |
| **-vv** | Prints out both the location of the Ansible configuration file and task results. |
| **-vvv** | Includes information about connections to managed hosts. |
| **-vvvv** | Includes additional verbosity such as scripts being executed, users connecting to managed hosts to execute scripts and so much more. |

**IMPLEMENTING MULTIPLE PLAYS**

A playbook can consist of multiple plays, not just one. Recall that a play is a collection of tasks to be executed. Each play can apply tasks to different managed hosts. Multiple plays come in handy in a complex deployment involving multiple tasks which need to be carried out on various hosts.

Writing a playbook with multiple plays is quite a simple and straightforward process. In the example below, we have 2 plays. The first installs Apache web server on a host whose IP is **173.82.56.50** while the second play instalsl the Mariadb database server on the host whose IP is **173.82.201.202**.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | ---  - name: play to setup web server    hosts: 173.82.56.50      tasks:    - name: Install Apache web server on CentOS 8      yum:        name: httpd        state: present    -  name: play to install mariadb     hosts: 173.82.201.202       tasks:     - name: Install Mariadb database server       yum:         name: mariadb         state: latest |

**REMOTE USERS AND PRIVILEGE ESCALATION IN PLAYS**

Other than what is specified in the default configuration file, Ansible Plays can include different remote users or privilege escalation settings for a play. These are defined in the play itself at the same level as the hosts or tasks keywords.

You can define the user that is executing the tasks in a playbook using the **remote\_user** keyword. This keyword also overrrides the user specified in the Ansible configuration file.



|  |  |
| --- | --- |
| 1 | <strong>remote\_user: user</strong> |

Additionally, if privilege escalation is enabled, then keywords such as **become\_user** become relevant.

**PRIVILEGE ESCALATION ATTRIBUTES**

We have tackled privilege escalation parameters before. In plays, you can use the boolean **become** parameter to either enable or disable privilege escalation. It can take either a Yes or No value as shown:



|  |  |
| --- | --- |
| 1 | become: true |

When privilege escalation is enabled, then the**become\_method** keyword comes in handy and is used to define the privilege escalation method to be used during a specific play. For instance, sudo is used for privilege escalation in our example below.



|  |  |
| --- | --- |
| 1 | become\_method: sudo |

Also, if privilege escalation is defined, then the become\_user can also be used to specify the user for privileges escalation in a specific play



|  |  |
| --- | --- |
| 1 | become\_user: privileged user |

**FINDING MODULES FOR TASKS**

As you implement Ansible playbooks, playing around with modules becomes routine task. Ansible ships with a wealth of modules that can be used in a myriad of applications including modules for use in cloud platforms such as AWS and Azure. In the previous chapter we had an overview of [Ansible modules](https://www.linuxteacher.com/buiding-an-ansible-inventory-rhce-ex294-exam-part-02/) and how they are used.

Let’s do some recap. You can list all the modules in Ansible, use the **ansible-doc**command. To view more information, append the **-l** flag to list module names and a summary of their functions.



|  |  |
| --- | --- |
| 1 | $ ansible-doc -l |

* A picture containing text, screenshot, font

  Description automatically generated

To display a particular module and its functions, use the **ansible-doc module-name** command. For example, to find out more information about the apt module, the command will be:



|  |  |
| --- | --- |
| 1 | $ ansible-doc apt |

The**-s**option give more insights into how a module can be used with ansible.



|  |  |
| --- | --- |
| 1 | $ ansible-doc -s  apt |

A screenshot of a computer

Description automatically generated with medium confidence

**PLAYBOOK SYNTAX VARIATIONS**

Lastly , let’s have a look at some of the syntax variations in Ansible YAML format.

**Comments**

Comments are lines of text that describe what a play or a task is all about. They start with a hash sign and are not executed at runtime.



|  |  |
| --- | --- |
| 1 | # This is a comment |

If there is content to the left of the comment, precede the number  
symbol with a space.



|  |  |
| --- | --- |
| 1 | remoteuser: user  # This is also a YAML comment |

**YAML Strings**

It’s not necesary to enclose strings in quotation marks even where there are spaces contained in the string. If you insist, you can enclose them in single or double quotation marks.



|  |  |
| --- | --- |
| 1  2  3  4  5 | An example of a string    'Yet another string'    "Now you get the picture" |

**YAML Lists**

Lists can be written in a normal single-dash style as shown:



|  |  |
| --- | --- |
| 1  2  3  4 | hosts:    - server1    - server2    - server3 |

Also, you can have the items in an inline format enclosed in square brackets as shown:



|  |  |
| --- | --- |
| 1 | hosts:  [server1, server2, server3] |

However, this should be avoided as it makes it harder to read.

**YAML dictionaries**

Dictionaries are key-value pairs that take the following format.



|  |  |
| --- | --- |
| 1  2 | name: httpd  state: present |

They can also be written in an inline-block format a shown, but this is quite clumsy and not recommended as you implement playbooks in Ansible.



|  |  |
| --- | --- |
| 1 | {name:httpd , state:present} |

**Playbook shorthand**

In some cases, some playbooks use the shorthand method of defining tasks by placing the key-value pairs on the same line as the module name:



|  |  |
| --- | --- |
| 1  2  3 | - tasks:      name: Install Apache web server      service: name=httpd enabled=true state=started |

While the playbook will run without any issues, this shorthand form is generally discouraged as it is hard to read and follow through.

Stacking the task’s keyword vertically is highly encouraged as it makes it easier to follow along as shown.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | - tasks:      name: Install Apache web server      service:        name:httpd        enabled:true        state:started |

And that’s it for this Chapter.