Understand Core Components of Ansible – Part 1

Ravi Saive | Last Updated: December 23, 2019 | Read Time: 5 mins | Ansible | 5 Comments

The Red Hat Certified Specialist in Ansible Automation exam (EX407) is a new certification program by Red Hat that tests your skills to use Ansible to automate the configuration of systems and applications.

The series will be titled "The Red Hat Certified Specialist in Ansible Automation exam (EX407)" and covers the following exam objectives based on Red Hat Enterprise Linux 7.5 and Ansible 2.7, which we will going to cover in this Ansible series:

To view fees and register for an exam in your country, check the Ansible Automation exam page.

> Part 10: How to Use Ansible Vault in Playbooks to Protect Sensitive Data

In this **Part 1** of the **Ansible** series, we will discuss some basic overview of core components in **Ansible**.

Understand Core Components of Ansible

Ansible is a free and opensource automation platform by **RedHat** that enables you to manage and control multiple servers from one central location. It's especially ideal when you have multiple and repetitive tasks that need to be performed. So instead of logging into each of these remote nodes and carrying out your tasks, you can comfortably do so from a central location and comfortably manage your servers.

This is beneficial when you want to maintain consistency in application deployment, reduce human error and automating repetitive and somewhat mundane tasks.

Of course, there are other alternatives to **Ansible** such as **Puppet**, **Chef**, and **Salt**. However, **Ansible** is mostly preferred because it is easy to use and simple to learn.

Why is it simple to learn you might ask? This is because **Ansible** uses **YAML** (**Yet Another Markup Language**) in its configuration and automation jobs which are human-readable and quite easy to follow. **YAML** uses **SSH** protocol to communicate with remote servers, unlike other automation platforms that require to you install an agent on remote nodes to communicate with them.

Before we get started with **Ansible**, it's important that you get acquainted with some basic terminologies so that you don't get lost or confused as we move forward.

Inventory

An **inventory** is a text file that contains a list of servers or nodes that you are managing and configuring. Usually, the servers are listed based on their hostnames or IP addresses.

An **inventory** file can contain remote systems defined by their IP addresses as shown:

```
10.200.50.50
10.200.50.51
10.200.50.52
```

Alternatively, they can be listed according to groups. In the example below, we have servers placed under 2 groups – **web servers** and **databases**. This way they can be referenced according to their group names and not their IP addresses. This further simplifies operation processes.

```
[webservers]
10.200.50.60
10.200.50.61

[databases]
10.200.50.70
10.200.50.71
```

You can have multiple groups with multiple servers if you are in a large production environment.

Playbook

A **playbook** is a set of configuration management scripts that define how tasks are to be executed on remote hosts or a group of host machines. The scripts or instructions are written in YAML format.

For instance, you can have a **playbook** file to install the Apache webserver on CentOS 7 and call it **httpd.yml**.

To create the **playbook** run the command.

```
$ touch playbook_name.yml
```

For example to create a playbook called **httpd**, run the command.

```
$ touch httpd.yml
```

A YAML file begins with 3 hyphens as shown. Inside the file, add the following instructions.

```
---
- name: This installs and starts Apache webserver
  hosts: webservers

  tasks:
  - name: Install Apache Webserver
    yum:   name=httpd  state=latest

 - name: check httpd status
    service:   name=httpd  state=started
```

The above playbook installs Apache web server on remote systems defined as webservers in the inventory file. After the installation of the webserver, Ansible later checks if the Apache web server is started and running.

Modules

Modules are discrete units of code used in playbooks for executing commands on remote hosts or servers. Each module is followed by an argument.

The basic format of a module is key: value.

```
- name: Install apache packages
    yum:   name=httpd  state=present
```

In the above YAML code snippet, -name and yum are modules.

Plays

An ansible play is a script or an instruction that defines the task to be carried out on a server. A collection of plays constitute a playbook. In other words, a playbook is a collection

of multiple plays, each of which clearly stipulates the task to be carried out on a server. Plays exist in YAML format.

Variables

If you have a background in programming, then most likely you have used variables. Basically, a variable represents a value. A variable can include letters, numerals, and underscores but MUST always begin with letters.

Variables are used when instructions vary from one system to another. This is especially true during the configuration or various services and features.

There are 3 main types of variables:

- Playbook variables
- Inventory variables
- Special variables

In **Ansible**, variables are first defined using the **vars k**, then followed by the variable name and the value.

The syntax is as shown:

```
vars:
Var name1: ‘My first variable’
        Var name2:  ‘My second variable’
```

Consider the code below.

```
- hosts: webservers
  vars:
    - web_directory:/var/www/html/
```

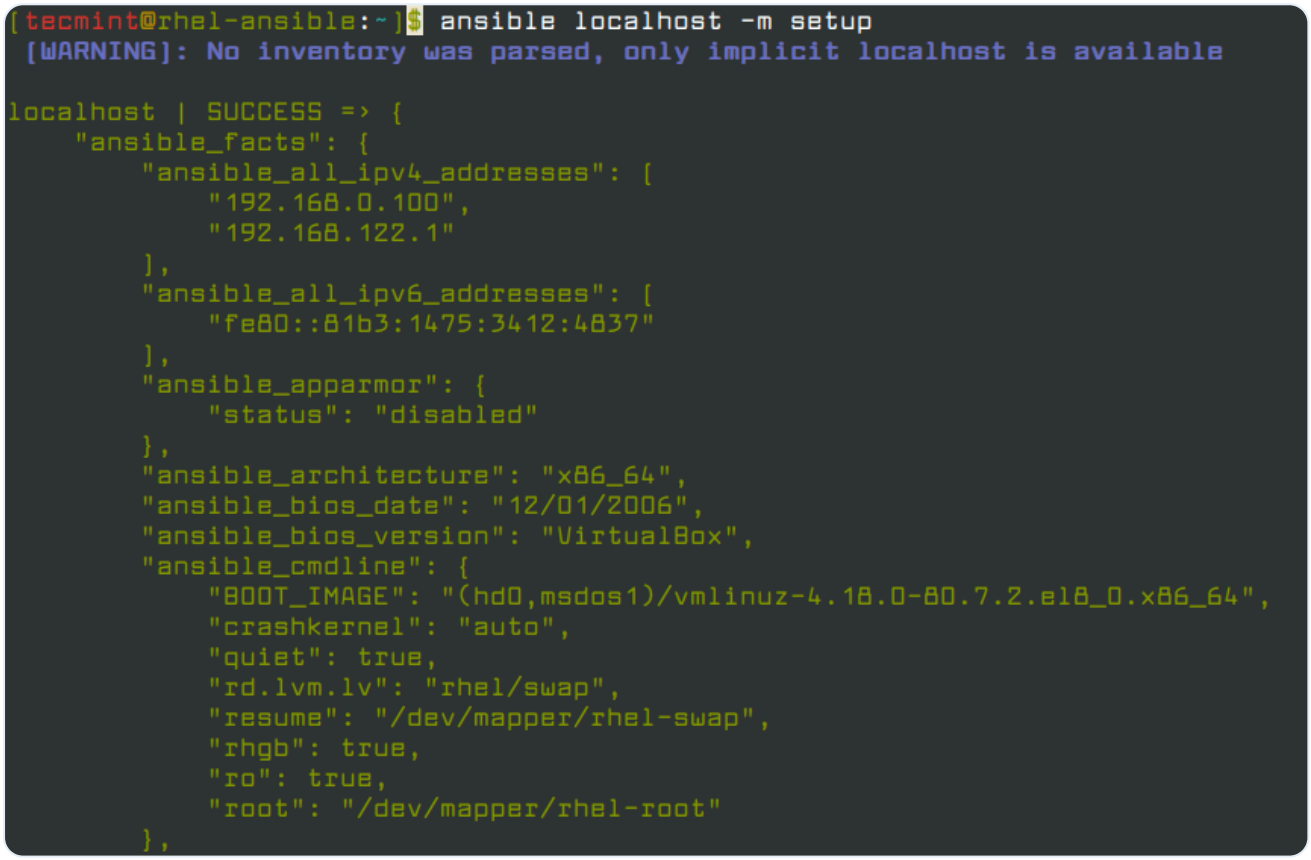
In the example above, the variable here is **web_directory** and it instructs ansible to create a directory in the **/var/www/html/** path.

Facts

Facts are system properties gathered by Ansible when it executes a playbook on a host system. The properties include hostname, OS family, CPU type, and CPU cores to mention a few.

To have a glimpse of the number of facts available for use issue the command.

```
$ ansible localhost -m setup
```



List Available Ansible Facts

As you can see, a huge number of facts have been displayed by default. You can further narrow down the results using the filter parameter as shown.

```
$ ansible localhost -m setup -a "filter=*ipv4"
```

```
[tecmint@rhel-ansible:~]$ ansible localhost -m setup -a "filter=*ipv4"
 [WARNING]: No inventory was parsed, only implicit localhost is available

localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "192.168.0.100",
            "alias": "enp0s3",
            "broadcast": "192.168.0.255",
            "gateway": "192.168.0.1",
            "interface": "enp0s3",
            "macaddress": "08:00:27:fc:a0:38",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.168.0.0",
            "type": "ether"
        }
    },
    "changed": false
}
[tecmint@rhel-ansible:~]$ _
```

List IPv4 Ansible Facts

Configuration Files

In Ansible, a configuration file is a file that contains different parameter settings that determine how Ansible runs. The default configuration file is the ansible.cfg file located in /etc/ansible/ directory.

You can view the configuration file by running:

```
$ cat /etc/ansible/ansible.cfg
```

```
[root@rhel-ansible:~/ansible-2.7.0]# cat /etc/ansible/ansible.cfg
# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml
#forks          = 5
#poll_interval  = 15
#sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
#transport      = smart
#remote_port    = 22
#module_lang    = C
#module_set_locale = False
```

Ansible Configuration File

As you can observe, several parameters are included such as **inventory** and **library** file paths, sudo user, plugin filters, modules, etc. These parameters can be adjusted simply by commenting them out and modifying the values therein.

Additionally, you can have multiple configurations files working with Ansible apart from your default config file.

Summary

Having looked at the core components in **Ansible**, we hope you are in a position to keep them at your fingertips and pick them out as we move forward. Join us on your next topic.

Ansible Exam Guide, Ansible Tips

Hey TecMint readers,

Exciting news! Every month, our top blog commenters will have the chance to win fantastic rewards, like free Linux eBooks such as RHCE, RHCSA, LFCS, Learn Linux, and Awk, each worth $20!

Learn more about the contest and stand a chance to win by sharing your thoughts below!

PREVIOUS ARTICLE:

Setup a Centralized Log Server with Rsyslog in CentOS/RHEL 8

NEXT ARTICLE:

How to Install Fail2Ban to Protect SSH on CentOS/RHEL 8

Ravi Saive

I am an experienced GNU/Linux expert and a full-stack software developer with over a decade in the field of Linux and Open Source technologies

Each tutorial at TecMint is created by a team of experienced Linux system administrators so that it meets our high-quality *standards*.

Related Posts

```
aaronk@tecmint:~$ ansible prod_servers -a "systemctl status firewalld" -u root
[redacted].235 | FAILED! => {
    "changed": false,
    "module_stderr": "Shared connection to [redacted].235 closed.\r\n",
    "module_stdout": "/bin/sh: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "rc": 127
}
[redacted]80 | FAILED! => {
    "changed": false,
    "module_stderr": "Shared connection to [redacted]80 closed.\r\n",
    "module_stdout": "/bin/sh: /usr/bin/python: No such file or directory\r\n",
    "msg": "MODULE FAILURE",
    "rc": 127
}
aaronk@tecmint:~$
```

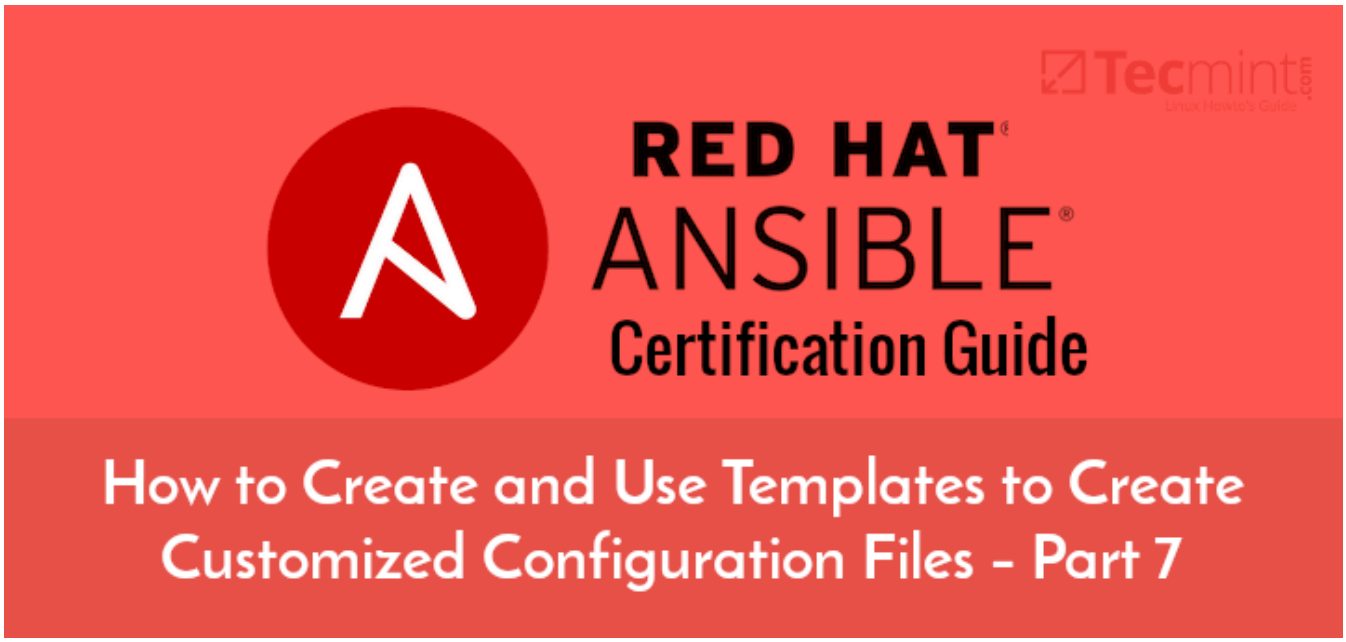
How to Fix “Shared connection to x.x.xx closed” Ansible Error

Tecmint’s Guide to RedHat Ansible Automation Exam Preparation Guide

How to Use Ansible Vault in Playbooks to Protect Sensitive Data – Part 10

How to Create and Download Roles on Ansible Galaxy and Use Them – Part 9

How to Work with Ansible Variables and Facts – Part 8



How to Create Templates in Ansible to Create Configurations On Managed Nodes – Part 7

5 Comments

Leave a Reply

Ahmed

November 22, 2019 at 3:28 pm

Please Publish Part5 onward tutorials.

Reply

Author

Ravi Saive

November 25, 2019 at 10:09 am

@Ahmed,

Yes, we are in progress to publish Part 6 of this Ansible series soon.

Reply

Author

Ravi Saive

December 6, 2019 at 12:51 pm

@Ahmed,

We have published Part 5 and Part 6 of this Ansible series, hope you like it.

Reply

Bernard

October 10, 2019 at 10:21 pm

Excellent comprehensive Ansible tutorial!

There's a, probably not intended; little error in your playbook example.

```
 tasks:
 - name: Install Apache Webserver
   yum:   name=httpd  state=present

- name: check httpd status
   yum:   name=httpd  state=started   <--- Error
```

Yum doesn't have a started option under it's state parameter, the service module does though.

YUM https://docs.ansible.com/ansible/latest/modules/yum_module.html

Service https://docs.ansible.com/ansible/latest/modules/service_module.html

Reply

Author

Ravi Saive

October 11, 2019 at 9:23 am

@Bernard,

Thanks for pointing out the error, corrected in the article.

Reply

Got Something to Say? Join the Discussion...

Thank you for taking the time to share your thoughts with us. We appreciate your decision to leave a comment and value your contribution to the discussion. It's important to note that we moderate all comments in accordance with our comment policy to ensure a respectful and constructive conversation.

Rest assured that your email address will remain private and will not be published or shared with anyone. We prioritize the privacy and security of our users.

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search...

Do You Enjoy My Blog?

Support from readers like YOU keeps this blog running. Buying me a cup of coffee is a simple and affordable way to show your appreciation and help keep the posts coming!

Buy Me a Coffee

Linux Commands and Tools

- How to Use IP Command in Linux [24 Useful Examples]
- How to Find Out Who is Using a File in Linux
- How to Record and Replay Linux Terminal Sessions
- Gdu – A Pretty Fast Disk Usage Analyzer for Linux
- How to Create and Execute a .Jar File in Linux Terminal
- 8 Useful X-window (Gui Based) Linux Commands – Part I

Linux Server Monitoring Tools

- How to Monitor Performance Of CentOS 8/7 Server Using Netdata
- How to Create a Centralized Log Server with Rsyslog in CentOS/RHEL 7
- Sysmon – A Graphical System Activity Monitor for Linux
- Configure Collectd as a Central Monitoring Server for Clients
- How to Install Nagios 4 in Ubuntu and Debian
- 9 Useful Commands to Get CPU Information on Linux

Learn Linux Tricks & Tips

- How to Repair and Defragment Linux System Partitions and Directories
- 2 Ways to Re-run Last Executed Commands in Linux
- Display Command Output or File Contents in Column Format

How to Find Out List of All Open Ports in Linux

10 Practical Examples Using Wildcards to Match Filenames in Linux

Learn Why 'less' is Faster Than 'more' Command for Effective File Navigation

Best Linux Tools

8 Useful Linux Security Features and Tools for Beginners

Top 5 Open Source Collaboration Platforms for Linux in 2024

20 Useful Security Features and Tools for Linux Admins

10 Tools to Take or Capture Desktop Screenshots in Linux

12 Best Media Server Software for Linux in 2024

18 Best NodeJS Frameworks for App Development in 2023