



Red Hat Ansible Automation Platform 2.4

Getting started with Ansible Playbooks

Getting started with ansible playbooks

Red Hat Ansible Automation Platform 2.4 Getting started with Ansible Playbooks

Getting started with ansible playbooks

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Learn how to set up an ansible playbook.

Table of Contents

PREFACE	3
MAKING OPEN SOURCE MORE INCLUSIVE	4
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. INTRODUCTION	6
1.1. HOW DO ANSIBLE PLAYBOOKS WORK?	6
1.2. HOW DO I USE ANSIBLE PLAYBOOKS?	7
1.2.1. From the CLI	8
1.2.2. From within the platform	8
1.3. STARTING AUTOMATION WITH ANSIBLE	8
1.4. BUILDING AN INVENTORY	8
1.4.1. Inventories in INI or YAML format	9
1.4.2. Tips for building inventories	10
1.4.3. Use metagroups	10
1.5. CREATE VARIABLES	11
1.6. CREATING YOUR FIRST PLAYBOOK	11
CHAPTER 2. USE A PLAYBOOK TO ESTABLISH A CONNECTION TO A MANAGED NODE	14
2.1. RUN A NETWORK ANSIBLE COMMAND	14
2.2. RUNNING A NETWORK ANSIBLE PLAYBOOK	14
2.3. GATHER FACTS FROM NETWORK DEVICES	18
CHAPTER 3. A PRACTICAL EXAMPLE OF AN ANSIBLE PLAYBOOK	20
3.1. PLAYBOOK EXECUTION	20

PREFACE

Thank you for your interest in Red Hat Ansible Automation Platform. Ansible Automation Platform is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.

This guide provides an introduction to the use of Ansible Playbooks..

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

If you have a suggestion to improve this documentation, or find an error, please contact technical support at <https://access.redhat.com> to create an issue on the Ansible Automation Platform Jira project using the **docs-product** component.

CHAPTER 1. INTRODUCTION

An Ansible Playbook is a blueprint for automation tasks, which are actions executed with limited manual effort across an inventory of solutions. Playbooks tell Ansible what to do on which devices. Instead of manually applying the same action to hundreds or thousands of similar technologies across IT environments, executing a playbook automatically completes the same action for the specified type of inventory—such as a set of routers.

Playbooks are regularly used to automate IT infrastructure—such as operating systems and Kubernetes platforms—networks, security systems, and code repositories like GitHub. You can use playbooks to program applications, services, server nodes, and other devices, without the manual overhead of creating everything from scratch. Playbooks, and the conditions, variables, and tasks within them, can be saved, shared, or reused indefinitely. This makes it easier for you to codify operational knowledge and ensure that the same actions are performed consistently.

1.1. HOW DO ANSIBLE PLAYBOOKS WORK?

Ansible Playbooks are lists of tasks that automatically execute for your specified inventory or groups of hosts. One or more Ansible tasks can be combined to make a play, that is, an ordered grouping of tasks mapped to specific hosts.

Tasks are executed in the order in which they are written.

A playbook can include one or more plays.

A playbook is composed of one or more **plays** in an ordered list.

The terms **playbook** and **play** are sports analogies.

Each play executes part of the overall goal of the playbook, running one or more tasks.

Each task calls an Ansible module.

Playbook

A list of plays that define the order in which Ansible performs operations, from top to bottom, to achieve an overall goal.

Play

An ordered list of tasks that maps to managed nodes in an inventory.

Task

A reference to a single module that defines the operations that Ansible performs.

Roles

Roles are a way to make code in playbooks reusable by putting the functionality into "libraries" that can then be used in any playbook as needed.

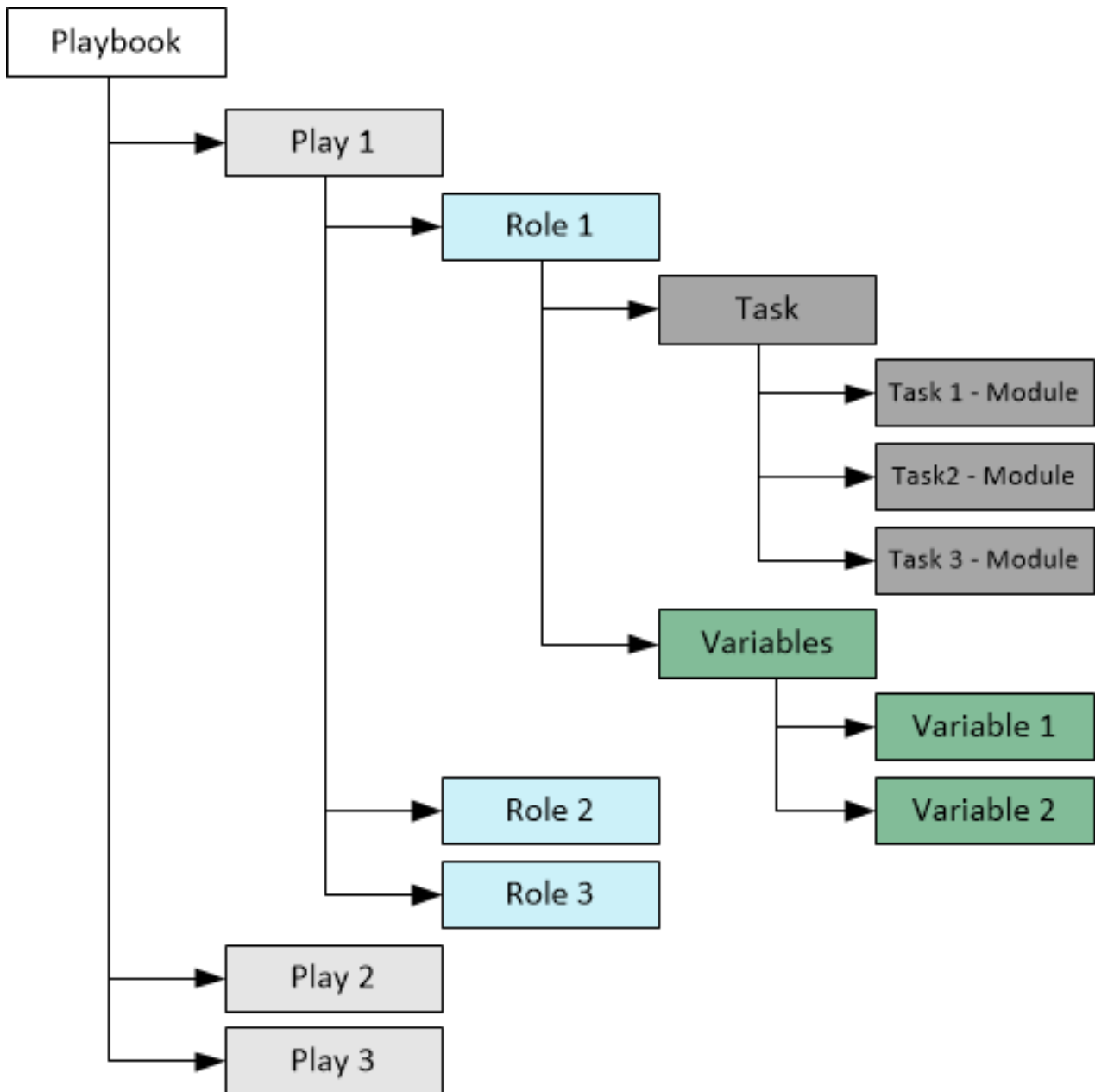
Module

A unit of code or binary that Ansible runs on managed nodes.

Ansible modules are grouped in collections with a *Fully Qualified Collection Name* (FQCN) for each module. Tasks are executed by modules, each of which performs a specific task in a playbook. A module contains metadata that determines when and where a task is executed, as well as which user executes it. There are thousands of Ansible modules that perform all kinds of IT tasks, such as:

- Cloud management

- User management
- Networking
- Security
- Configuration management
- Communication



1.2. HOW DO I USE ANSIBLE PLAYBOOKS?

Ansible uses the YAML syntax. YAML is a human-readable language that enables you to create playbooks without having to learn a complicated coding language.

For more information on YAML, see [YAML Syntax](#) and consider installing an add-on for your text editor, see [Other Tools and Programs](#) to help you write clean YAML syntax in your playbooks.

There are two ways of using Ansible Playbooks:

- From the *command line interface* (CLI)
- Using Red Hat Ansible Automation Platform’s push-button deployments.

1.2.1. From the CLI

After installing the open source Ansible project or Red Hat Ansible Automation Platform by using

```
$ sudo dnf install ansible
```

in the Red Hat Enterprise Linux CLI, you can use the **ansible-playbook** command to run Ansible Playbooks.

1.2.2. From within the platform

The Red Hat Ansible Automation Platform user interface offers push-button Ansible Playbook deployments that can be used as part of larger jobs or job templates. These deployments come with additional safeguards that are particularly helpful to users who are newer to IT automation, or those without as much experience working in the CLI.

1.3. STARTING AUTOMATION WITH ANSIBLE

Get started with Ansible by creating an automation project, building an inventory, and creating a **Hello World** playbook.

Prerequisites

- The Ansible package must be installed.

Procedure

- Create a project folder on your filesystem.

```
mkdir ansible_quickstart  
cd ansible_quickstart
```

Using a single directory structure makes it easier to add to source control, and reuse and share automation content.

1.4. BUILDING AN INVENTORY

Inventories organize managed nodes in centralized files that provide Ansible with system information and network locations. Using an inventory file, Ansible can manage a large number of hosts with a single command. To complete the following steps, you need the IP address or fully qualified domain name (FQDN) of at least one host system. For demonstration purposes, the host could be running locally in a container or a virtual machine.

You must also ensure that your public SSH key is added to the **authorized_keys** file on each host. Use the following procedure to build an inventory.

Procedure

Create a file named **inventory.ini** in the **ansible_quickstart** directory that you created. Add a new **[myhosts]** group to the inventory.ini file and specify the IP address or fully qualified domain name (FQDN) of each host system.

```
[myhosts]
192.0.2.50
192.0.2.51
192.0.2.52
```

Verify your inventory, using:

```
ansible-inventory -i inventory.ini --list
```

Ping the **myhosts** group in your inventory, using:

```
ansible myhosts -m ping -i inventory.ini
```

Pass the **-u** option with the Ansible command if the username is different on the control node and the managed node(s).

```
192.0.2.50 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.0.2.51 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
192.0.2.52 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

You have successfully built an inventory.

1.4.1. Inventories in INI or YAML format

You can create inventories in either INI files or in YAML. In most cases, such as the preceding example, INI files are straightforward and easy to read for a small number of managed nodes. Creating an inventory in YAML format becomes a sensible option as the number of managed nodes increases.

For example, the following is an equivalent of the **inventory.ini** that declares unique names for managed nodes and uses the **ansible_host** field:

```
myhosts:
```

```
hosts:
  my_host_01:
    ansible_host: 192.0.2.50
  my_host_02:
    ansible_host: 192.0.2.51
  my_host_03:
    ansible_host: 192.0.2.52
```

1.4.2. Tips for building inventories

- Ensure that group names are meaningful and unique.
- Group names are also case sensitive.
- Do not use spaces, hyphens, or preceding numbers (use floor_19, not 19th_floor) in group names.
- Group hosts in your inventory logically according to their What, Where, and When:
 - What: Group hosts according to the topology, for example: db, web, leaf, spine.
 - Where: Group hosts by geographic location, for example: datacenter, region, floor, building.
 - When: Group hosts by stage, for example: development, test, staging, production.

1.4.3. Use metagroups

Create a metagroup that organizes multiple groups in your inventory with the following syntax:

```
metagroupname:
  children:
```

The following inventory illustrates a basic structure for a data center. This example inventory contains a network metagroup that includes all network devices and a datacenter metagroup that includes the network group and all webservers.

```
leafs:
  hosts:
    leaf01:
      ansible_host: 192.0.2.100
    leaf02:
      ansible_host: 192.0.2.110

spines:
  hosts:
    spine01:
      ansible_host: 192.0.2.120
    spine02:
      ansible_host: 192.0.2.130

network:
  children:
    leafs:
    spines:
```

```
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
    webserver02:
      ansible_host: 192.0.2.150

datacenter:
  children:
    network:
      webservers:
```

1.5. CREATE VARIABLES

Variables set values for managed nodes, such as the IP address, FQDN, operating system, and SSH user, so you do not need to pass them when running Ansible commands.

Variables can apply to specific hosts.

```
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
```

Variables can also apply to all hosts in a group.

```
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
  vars:
    ansible_user: my_server_user
```

For more information about inventories and Ansible inventory variables, see [About the Installer Inventory file](#) and [Inventory file variables](#).

1.6. CREATING YOUR FIRST PLAYBOOK

Use the following procedure to create a playbook that pings your hosts and prints a “Hello world” message.

Procedure

1. Create a file named **playbook.yaml** in your **ansible_quickstart** directory, with the following content:

```

- name: My first play
  hosts: myhosts
  tasks:
    - name: Ping my hosts
      ansible.builtin.ping:

    - name: Print message
      ansible.builtin.debug:
        msg: Hello world

```

2. Run your playbook, using the following command:

```
ansible-playbook -i inventory.ini playbook.yaml
```

3. Ansible returns the following output:

```

PLAY [My first play] *****

TASK [Gathering Facts] *****
ok: [192.0.2.50]
ok: [192.0.2.51]
ok: [192.0.2.52]

TASK [Ping my hosts] *****
ok: [192.0.2.50]
ok: [192.0.2.51]
ok: [192.0.2.52]

TASK [Print message] *****
ok: [192.0.2.50] => {
  "msg": "Hello world"
}
ok: [192.0.2.51] => {
  "msg": "Hello world"
}
ok: [192.0.2.52] => {
  "msg": "Hello world"
}

PLAY RECAP *****
192.0.2.50: ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
192.0.2.51: ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
192.0.2.52: ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0

```

In this output you can see:

- The names that you give the play and each task. Always use descriptive names that make it easy to verify and troubleshoot playbooks.
- The Gather Facts task runs implicitly. By default Ansible gathers information about your inventory that it can use in the playbook.
- The status of each task. Each task has a status of **ok** which means it ran successfully.

- The play recap that summarizes results of all tasks in the playbook per host. In this example, there are three tasks so **ok=3** indicates that each task ran successfully.

CHAPTER 2. USE A PLAYBOOK TO ESTABLISH A CONNECTION TO A MANAGED NODE

To confirm your credentials, you can connect to a network device manually and retrieve its configuration. Replace the sample user and device name with your real credentials.

For example, for a VyOS router:

```
ssh my_vyos_user@vyos.example.net
show config
exit
```

2.1. RUN A NETWORK ANSIBLE COMMAND

Instead of manually connecting and running a command on the network device, you can retrieve its configuration with a single Ansible command.

```
ansible all -i vyos.example.net, -c ansible.netcommon.network_cli -u \
my_vyos_user -k -m vyos.vyos.vyos_facts -e \
ansible_network_os=vyos.vyos.vyos
```

The flags in this command set seven values:

- the host group(s) to which the command should apply (in this case, **all**)
- the inventory (**-i**, the device or devices to target - without the trailing comma **-i** points to an inventory file)
- the connection method (**-c**, the method for connecting and executing ansible)
- the user (**-u**, the username for the SSH connection)
- the SSH connection method (**-k**, prompt for the password)
- the module (**-m**, the Ansible module to run, using the fully qualified collection name (FQCN))
- an extra variable (**-e**, in this case, setting the network OS value)



NOTE

If you use **ssh-agent** with ssh keys, Ansible loads them automatically. You can omit the **-k** flag.

If you are running Ansible in a virtual environment, you must also add the variable **ansible_python_interpreter=/path/to/venv/bin/python**.

2.2. RUNNING A NETWORK ANSIBLE PLAYBOOK

If you want to run a particular command every day, you can save it in a playbook and run it with **ansible-playbook** instead of **ansible**. The playbook can store a lot of the parameters you provided with flags at the command line, leaving less to type at the command line. You need two files for this, a playbook and an inventory file.

Prerequisites

Download **first_playbook.yml** from [here](#).

The playbook looks like this:

```
---
- name: Network Getting Started First Playbook
  connection: ansible.netcommon.network_cli
  gather_facts: false
  hosts: all
  tasks:
    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all
    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_version }}"
```

Label	Description
gather_facts	Ansible's native fact gathering (ansible.builtin.setup) is disabled here because the playbook relies on the facts provided by a platform-specific module (vyos.vyos.vyos_facts) in this networking collection.

The playbook sets three of the seven values from the command line above:

- the group (**hosts: all**)
- the connection method (**connection: ansible.netcommon.network_cli**) and
- the module (in each task).

With those values set in the playbook, you can omit them on the command line. The playbook also adds a second task to show the configuration output.

When facts are gathered from a system, either through a collection-specific fact module such as **vyos.vyos.vyos_facts** or **ansible.builtin.setup**, the gathered data is held in memory for use by future tasks instead of being written to the console.

When a module runs in a playbook, the output is held in memory for use by future tasks instead of written to the console. With most other modules you must explicitly register a variable to store and reuse the output of a module or task.

For more information about facts, see [Ansible facts] in the *Ansible Playbook Reference Guide*.

The following debug task lets you see the results in your shell.

Procedure

1. Run the playbook with the following command.

```
ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook.yml
```

The playbook contains one play with two tasks, and generates output like this.

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook.yml

PLAY [Network Getting Started First Playbook]
*****

TASK [Get config for VyOS devices]
*****

ok: [vyos.example.net]

TASK [Display the config]
*****

ok: [vyos.example.net] => {
  "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}
```

- Now that you can retrieve the device configuration, you can try updating it with Ansible.
- Download **first_playbook_ext.yml** from [here](#), which is an extended version of the first playbook: The playbook looks like this:

```
---

- name: Network Getting Started First Playbook Extended
  connection: ansible.netcommon.network_cli
  gather_facts: false
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_version
        }}"

    - name: Update the hostname
      vyos.vyos.vyos_config:
        backup: yes
        lines:
          - set system host-name vyos-changed

    - name: Get changed config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all
```

```
- name: Display the changed config
  debug:
    msg: "The new hostname is {{ ansible_net_hostname }} and the OS is {{
ansible_net_version }}"
```

4. The extended first playbook has five tasks in a single play.

5. Run the playbook with the following command.

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook_ext.yml
```

6. The output shows you the change Ansible made to the configuration:

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook_ext.yml
```

```
PLAY [Network Getting Started First Playbook Extended]
```

```
*****
*****
```

```
TASK [Get config for VyOS devices]
```

```
*****
*****
```

```
ok: [vyos.example.net]
```

```
TASK [Display the config]
```

```
*****
*****
```

```
ok: [vyos.example.net] => {
  "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}
```

```
TASK [Update the hostname]
```

```
*****
*****
```

```
changed: [vyos.example.net]
```

```
TASK [Get changed config for VyOS devices]
```

```
*****
*****
```

```
ok: [vyos.example.net]
```

```
TASK [Display the changed config]
```

```
*****
*****
```

```
ok: [vyos.example.net] => {
  "msg": "The new hostname is vyos-changed and the OS is VyOS 1.1.8"
}
```

```
PLAY RECAP
```

```
*****
*****
```

```
vyos.example.net      : ok=5  changed=1  unreachable=0  failed=0
```

2.3. GATHER FACTS FROM NETWORK DEVICES

The **gather_facts** keyword supports gathering network device facts in standardized key/value pairs. You can feed these network facts into further tasks to manage the network device. You can also use the **gather_network_resources** parameter with the network ***_facts** modules (such as **arista.eos.eos_facts**) to return a subset of the device configuration, as shown below.

```
- hosts: arista
gather_facts: True
gather_subset: interfaces
module_defaults:
  arista.eos.eos_facts:
    gather_network_resources: interfaces
```

The playbook returns the following interface facts:

```
"network_resources": {
  "interfaces": [
    {
      "description": "test-interface",
      "enabled": true,
      "mtu": "512",
      "name": "Ethernet1"
    },
    {
      "enabled": true,
      "mtu": "3000",
      "name": "Ethernet2"
    },
    {
      "enabled": true,
      "name": "Ethernet3"
    },
    {
      "enabled": true,
      "name": "Ethernet4"
    },
    {
      "enabled": true,
      "name": "Ethernet5"
    },
    {
      "enabled": true,
      "name": "Ethernet6"
    }
  ]
}
```



NOTE

gather_network_resources renders configuration data as facts for all supported resources (**interfaces/bgp/ospf/etc**), whereas **gather_subset** is primarily used to fetch operational data.

You can store these facts and use them directly in another task, such as with the [eos_interfaces](#) resource module.

CHAPTER 3. A PRACTICAL EXAMPLE OF AN ANSIBLE PLAYBOOK

Ansible can communicate with many different device classifications, from cloud-based REST APIs, to Linux and Windows systems, networking hardware, and much more.

The following is a sample of two Ansible modules automatically updating two types of servers.

3.1. PLAYBOOK EXECUTION

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple 'plays' can orchestrate multi-machine deployments, running one play on your webservers, then another play on your database servers, then a third play on your network infrastructure, and so on.

At a minimum, each play defines two things:

- the managed nodes to target, using a pattern
- at least one task to execute

In Ansible 2.10 and later, use the fully-qualified collection name in your playbooks to ensure the correct module is selected, because multiple collections can contain modules with the same name (for example, **user**).

For further information, see [Using collections in a playbook](#) .

In this example, the first play targets the web servers; the second play targets the database servers.

```
---
- name: Update web servers
  hosts: webservers
  become: true

  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest
    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
        mode: "0644"

- name: Update db servers
  hosts: databases
  become: true

  tasks:
    - name: Ensure postgresql is at the latest version
      ansible.builtin.yum:
        name: postgresql
        state: latest
    - name: Ensure that postgresql is started
```



```
ansible.builtin.service:  
  name: postgresql  
  state: started
```

The playbook contains two plays:

- The first checks if the web server software is up to date and runs the update if necessary.
- The second checks if database server software is up to date and runs the update if necessary.

Your playbook can include more than just a hosts line and tasks.

For example, this example playbook sets a `remote_user` for each play. This is the user account for the SSH connection. You can add other Playbook Keywords at the playbook, play, or task level to influence how Ansible behaves. Playbook keywords can control the connection plugin, whether to use privilege escalation, how to handle errors, and more.

To support a variety of environments, Ansible enables you to set many of these parameters as command-line flags, in your Ansible configuration, or in your inventory. Learning the precedence rules for these sources of data can help you as you expand your Ansible ecosystem