Learning Ansible with Rocky (English version)

A book from the Documentation Team

Version : 2025/07/02

Rocky Documentation Team

Copyright © 2023 The Rocky Enterprise Software Foundation

Table of contents

1. Licence	5
2. Learning Ansible with Rocky	6
3. Ansible Basics	7
3.1 The Ansible vocabulary	9
3.2 Installation on the management server	10
3.2.1 Installation from EPEL	10
3.2.2 Installation from python pip	11
3.3 Configuration files	12
3.3.1 The inventory file /etc/ansible/hosts	13
3.4 ansible command line usage	15
3.4.1 Preparing the client	17
3.4.2 Test with the ping module	18
3.5 Key authentication	19
3.5.1 Creating an SSH key	19
3.5.2 Private key authentication test	20
3.6 Using Ansible	20
3.6.1 The modules	20
3.6.2 Exercises	23
3.7 Playbooks	24
3.7.1 Example of Apache and MySQL playbook	25
3.8 Exercises results	29
4. Ansible Intermediate	31
4.1 The variables	31
4.1.1 Outsourcing variables	33
4.1.2 Display a variable	33
4.1.3 Save the return of a task	34
4.1.4 Exercises:	34
4.2 Loop management	35
4.2.1 Exercises:	37
4.3 Conditionals	37
4.3.1 Exercises:	39
4.4 Managing changes: the handlers	39
4.5 Asynchronous tasks	41
4.6 Exercise results	42

5. Ansible - Management of Files	49
5.1 ini_file module	49
5.2 lineinfile module	50
5.3 copy module	50
5.4 fetch module	51
5.5 template module	51
5.6 get_url module	52
6. Ansible Galaxy: Collections and Roles	53
6.1 ansible-galaxy command	53
6.2 Ansible Roles	54
6.2.1 Installing useful Roles	54
6.2.2 Introduction to Role development	58
6.2.3 Practical work: create a first simple role	59
6.3 Ansible Collections	64
6.3.1 Creating your own collection	66
7. Ansible Deployments with Ansistrano	67
7.1 Introduction	67
7.2 Labs	69
7.2.1 Deploying the Web server	69
7.2.2 Deploying the software	72
7.2.3 Checking on the server	74
7.2.4 Limit the number of releases	75
7.2.5 Using shared_paths and shared_files	76
7.2.6 Use a sub-directory of the repository for deployment	78
7.2.7 Managing git branch or tags	80
7.2.8 Actions between deployment steps	82
8. Ansible - Large Scale infrastructure	86
8.1 Variables storage	87
8.2 About Ansible tags	88
8.3 About the directory layout	89
8.4 Tests	91
8.5 Benefits	94
9. Ansible - Working with filters	95
9.1 Converting data	96
9.2 Join the elements of a list	99
9.3 Transforming dictionaries into lists (and vice versa)	100
9.4 Working with lists	101
9.5 Transformation json/yaml	103

9.6 Default values, optional variables, protect variables		
9.7 Associate a value according to another one (ternary)	104	
9.8 Some other filters	105	
10. Management server optimizations	106	
10.1 The ansible.cfg configuration file	107	
10.2 Caching the facts	109	
10.3 Using Vault	109	
10.4 Working with Windows servers	111	
10.5 Working with IP modules	112	
10.6 Generating a CMDB	112	

1. Licence

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

RockyLinux materials are published under Creative Commons-BY-SA. This means you are free to share and transform the material, while respecting the author's rights.

BY : **Attribution**. You must cite the name of the original author.

SA : Share Alike.

• Creative Commons-BY-SA licence : https://creativecommons.org/licenses/by-sa/ 4.0/

The documents and their sources are freely downloadable from:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Our media sources are hosted at github.com. You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using mkdocs. You will find instructions for generating your document here.

How can I contribute to the documentation project?

You'll find all the information you need to join us on our git project home page.

We wish you all a pleasant reading and hope you enjoy the content.

2. Learning Ansible with Rocky

Ansible is a simple, yet powerful, automation engine for Linux. This tutorial will guide you through the concepts of using Ansible to automate your IT tasks in a way that is (hopefully) fun and informative. Using the exercises throughout these chapters, will help you gain a comfort level with Ansible in real-world applications.

3. Ansible Basics

In this chapter you will learn how to work with Ansible.

Objectives: In this chapter you will learn how to:

✓ Implement Ansible;

 \checkmark Apply configuration changes on a server;

Create first Ansible playbooks;

🕅 ansible, module, playbook .

Knowledge: $\star \star \star$ Complexity: $\star \star$

Reading time: 30 minutes

Ansible centralizes and automates administration tasks. It is:

• agentless (it does not require specific deployments on clients),

• idempotent (same effect each time it is run).

It uses the **SSH** protocol to remotely configure Linux clients or the **WinRM** protocol to work with Windows clients. If none of these protocols is available, it is always possible for Ansible to use an API, which makes Ansible a real Swiss army knife for the configuration of servers, workstations, docker services, network equipment, etc. (almost everything in fact).

```
🛕 Warning
```

The opening of SSH or WinRM flows to all clients from the Ansible server, makes it a critical element of the architecture that must be carefully monitored.

As Ansible is mainly push-based, it will not keep the state of its targeted servers between each of its executions. On the contrary, it will perform new state checks each time it is executed. It is said to be stateless. It will help you with:

- provisioning (deploying a new VM),
- application deployments,
- configuration management,
- automation,
- orchestration (when more than 1 target is in use).

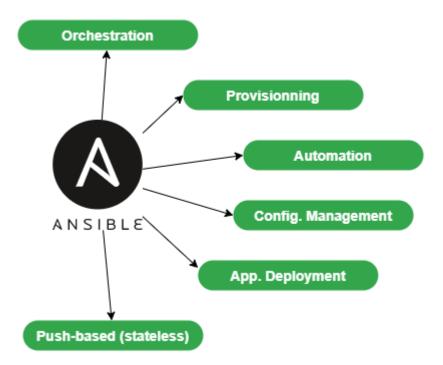
🖍 Note

Ansible was originally written by Michael DeHaan, the founder of other tools such as Cobbler.



The earliest first version was 0.0.1, released on March 9, 2012.

On October 17, 2015, AnsibleWorks (the company behind Ansible) was acquired by Red Hat for \$150 million.



To offer a graphical interface to your daily use of Ansible, you can install some tools like Ansible Tower (RedHat), which is not free, its opensource counterpart Awx, or other projects like Jenkins and the excellent Rundeck can also be used.

Abstract

To follow this training, you will need at least 2 servers under Rocky8:

• the first one will be the **management machine**, Ansible will be installed on it.

• the second one will be the server to configure and manage (another Linux than Rocky Linux will do just as well).

In the examples below, the administration station has the IP address 172.16.1.10, the managed station 172.16.1.11. It is up to you to adapt the examples according to your IP addressing plan.

3.1 The Ansible vocabulary

- The **management machine**: the machine on which Ansible is installed. Since Ansible is **agentless**, no software is deployed on the managed servers.
- The **managed nodes**: the target devices that Ansible manages are also referred to as "hosts." These can be servers, network appliances, or any other computer.
- The **inventory**: a file containing information about the managed servers.
- The **tasks**: a task is a block defining a procedure to be executed (e.g., create a user or a group, install a software package, etc.).

- A **module**: a module abstracts a task. There are many modules provided by Ansible.
- The **playbooks**: a simple file in yaml format defining the target servers and the tasks to be performed.
- A **role**: a role allows you to organize the playbooks and all the other necessary files (templates, scripts, etc.) to facilitate the sharing and reuse of code.
- A **collection**: a collection includes a logical set of playbooks, roles, modules, and plugins.
- The **facts**: these are global variables containing information about the system (machine name, system version, network interface and configuration, etc.).
- The **handlers**: these are used to cause a service to be stopped or restarted in the event of a change.

3.2 Installation on the management server

Ansible is available in the *EPEL* repository, but may sometimes be too old for the current version, and you'll want to work with a more recent version.

We will therefore consider two types of installation:

- the one based on EPEL repositories
- one based on the pip python package manager

The *EPEL* is required for both versions, so you can go ahead and install that now:

• EPEL installation:

sudo dnf install epel-release

3.2.1 Installation from EPEL

If we install Ansible from the *EPEL*, we can do the following:

sudo dnf install ansible

And then verify the installation:

```
$ ansible --version
ansible [core 2.14.2]
config file = /etc/ansible/ansible.cfg
configured module search path = ['/home/rocky/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python3.11/site-packages/ansible
ansible collection location = /home/rocky/.ansible/collections:/usr/share/
ansible/collections
executable location = /usr/bin/ansible
python version = 3.11.2 (main, Jun 22 2023, 04:35:24) [GCC 8.5.0 20210514
(Red Hat 8.5.0-18)] (/usr/bin/python3.11)
jinja version = 3.1.2
libyaml = True
$ python3 --version
Python 3.6.8
```

Please note that ansible comes with its own version of python, different from the system version of python (here 3.11.2 vs 3.6.8). You'll need to take this into account when pip-installing the python modules required for your installation (e.g. pip3.11 install PyVMomi).

3.2.2 Installation from python pip

As we want to use a newer version of Ansible, we will install it from python3-pip:



At this stage, we can choose to install ansible with the version of python we want.

sudo dnf install python38 python38-pip python38-wheel python3-argcomplete rust
cargo curl

🖍 Note

python3-argcomplete is provided by *EPEL*. Please install epel-release if not done yet. This package will help you complete Ansible commands.

We can now install Ansible:

```
pip3.8 install --user ansible
activate-global-python-argcomplete --user
```

Check your Ansible version:

```
$ ansible --version
ansible [core 2.13.11]
config file = None
configured module search path = ['/home/rocky/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
ansible python module location = /home/rocky/.local/lib/python3.8/site-
packages/ansible
ansible collection location = /home/rocky/.ansible/collections:/usr/share/
ansible/collections
executable location = /home/rocky/.local/bin/ansible
python version = 3.8.16 (default, Jun 25 2023, 05:53:51) [GCC 8.5.0 20210514
(Red Hat 8.5.0-18)]
jinja version = 3.1.2
libyaml = True
```

Note

The manually installed version in our case is older than the version packaged by RPM because we used an older version of python. This observation will vary with time and the age of the distribution and the python version of course.

3.3 Configuration files

The server configuration is located under /etc/ansible.

There are two main configuration files:

- The main configuration file ansible.cfg where the commands, modules, plugins, and ssh configuration reside;
- The client machine management inventory file hosts where the clients, and groups of clients are declared.

The configuration file would automatically be created if Ansible was installed with its RPM package. With a pip installation, this file does not exist. We'll have to create it by hand thanks to the ansible-config command:

Example:

ansible-config init --disabled > /etc/ansible/ansible.cfg

The --disabled option allows you to comment out the set of options by prefixing them with a ; .

🖍 Note

You can also choose to embed the ansible configuration in your code repository, with Ansible loading the configuration files it finds in the following order (processing the first file it encounters and ignoring the rest):

• if the environment variable \$ANSIBLE_CONFIG is set, load the specified file.

ansible.cfg if exists in the current directory.

• ~/.ansible.cfg if exists (in the user's home directory).

The default file is loaded if none of these three files are found.

3.3.1 The inventory file /etc/ansible/hosts

As Ansible will have to work with all your equipment to be configured, providing it with one (or more) well-structured inventory file(s) that perfectly matches your organization is essential.

It is sometimes necessary to think carefully about how to build this file.

Go to the default inventory file, which is located under /etc/ansible/hosts. Some examples are provided and commented:

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
```

```
#
#
   - Comments begin with the '#' character
   - Blank lines are ignored
#
   - Groups of hosts are delimited by [header] elements
#
    - You can enter hostnames or ip addresses
#
    - A hostname/ip can be a member of multiple groups
#
# Ex 1: Ungrouped hosts, specify before any group headers:
## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10
# Ex 2: A collection of hosts belonging to the 'webservers' group:
## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192,168,1,110
# If you have multiple hosts following a pattern, you can specify
# them like this:
## www[001:006].example.com
# Ex 3: A collection of database servers in the 'dbservers' group:
## [dbservers]
##
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57
# Here's another example of host ranges, this time there are no
# leading 0s:
## db-[99:101]-node.example.com
```

As you can see, the file provided as an example uses the INI format, which is well known to system administrators. Please note that you can choose another file format (like yaml for example), but for the first tests, the INI format is well adapted to our future examples. The inventory can be generated automatically in production, especially if you have a virtualization environment like VMware VSphere or a cloud environment (Aws, OpenStack, or another).

• Creating a hostgroup in /etc/ansible/hosts:

As you may have noticed, the groups are declared in square brackets. Then come the elements belonging to the groups. You can create, for example, a rocky8 group by inserting the following block into this file:

[rocky8] 172.16.1.10 172.16.1.11

Groups can be used within other groups. In this case, it must be specified that the parent group is composed of subgroups with the :children attribute like this:

```
[linux:children]
rocky8
debian9
[ansible:children]
ansible_management
ansible_clients
[ansible_management]
172.16.1.10
[ansible_clients]
172.16.1.10
```

We won't go any further on inventory, but if you are interested, consider checking this link.

Now that our management server is installed and our inventory is ready, it's time to run our first ansible commands.

3.4 ansible command line usage

The ansible command launches a task on one or more target hosts.

ansible <host-pattern> [-m module_name] [-a args] [options]

Examples:

🛕 Warning

Since we have not yet configured authentication on our 2 test servers, not all the following examples will work. They are given as examples to facilitate understanding, and will be fully functional later in this chapter.

• List the hosts belonging to the rocky8 group:

ansible rocky8 --list-hosts

• Ping a host group with the ping module:

ansible rocky8 -m ping

• Display facts from a host group with the setup module:

ansible rocky8 -m setup

• Run a command on a host group by invoking the command module with arguments:

ansible rocky8 -m command -a 'uptime'

• Run a command with administrator privileges:

ansible ansible_clients --become -m command -a 'reboot'

• Run a command using a custom inventory file:

ansible rocky8 -i ./local-inventory -m command -a 'date'

Note

As in this example, it is sometimes simpler to separate the declaration of managed devices into several files (by cloud project for example) and provide Ansible with the path to these files, rather than to maintain a long inventory file.

Option	Information
-a 'arguments'	The arguments to pass to the module.
-b -K	Requests a password and runs the command with higher privileges.
user=username	Uses this user to connect to the target host instead of the current user.
become-user=username	Executes the operation as this user (default: root).
- C	Simulation. Does not make any changes to the target but tests it to see what should be changed.
-m module	Runs the module called

3.4.1 Preparing the client

On both management machine and clients, we will create an ansible user dedicated to the operations performed by Ansible. This user will have to use sudo rights, so it will have to be added to the wheel group.

This user will be used:

- On the administration station side: to run ansible commands and SSH to managed clients.
- On the managed stations (here the server that serves as your administration station also serves as a client, so it is managed by itself) to execute the commands launched from the administration station: it must therefore have sudo rights.

On both machines, create an ansible user, dedicated to ansible:

```
sudo useradd ansible
sudo usermod -aG wheel ansible
```

Set a password for this user:

sudo passwd ansible

Modify the sudoers config to allow members of the wheel group to sudo without password:

sudo visudo

Our goal here is to comment out the default, and uncomment the NOPASSWD option so that these lines look like this when we are done:

```
## Allows people in group wheel to run all commands
# %wheel ALL=(ALL) ALL
## Same thing without a password
%wheel ALL=(ALL) NOPASSWD: ALL
```

🛕 Warning

If you receive the following error message when entering Ansible commands, it probably means that you forgot this step on one of your clients: "msg": "Missing sudo password

When using management from this point on, start working with this new user:

sudo su - ansible

3.4.2 Test with the ping module

By default, password login is not allowed by Ansible.

Uncomment the following line from the [defaults] section in the /etc/ansible/ ansible.cfg configuration file and set it to True:

ask_pass = True

Run a ping on each server of the rocky8 group:

```
# ansible rocky8 -m ping
SSH password:
172.16.1.10 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
172.16.1.11 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Note You are asked for the ansible password of the remote servers, which is a security problem...

🜢 Tip

If you get this error "msg": "to use the 'ssh' connection type with passwords, you must install the sshpass program", you can just install sshpass on the management station:

\$ sudo dnf install sshpass

Abstract

You can now test the commands that didn't work previously in this chapter.

3.5 Key authentication

Password authentication will be replaced by a much more secure private/public key authentication.

3.5.1 Creating an SSH key

The dual-key will be generated with the command ssh-keygen on the management station by the ansible user:

```
[ansible]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:0a1d2hYzzd00e/K10XPad25TA1nrSVRPIuS4fnmKr9g
ansible@localhost.localdomain
The key's randomart image is:
+---[RSA 3072]----+
            .0.+
            0.=.|
           . . + +|
          0. = =.|
         S \circ = B \cdot \circ |
          = + = = + |
         + = 0 + B
```

```
| 0 + 0 *@|
| . Eo0 .+B|
+----[SHA256]----+
```

The public key can be copied to the servers:

```
# ssh-copy-id ansible@172.16.1.10
# ssh-copy-id ansible@172.16.1.11
```

Re-comment the following line from the [defaults] section in the /etc/ansible/ ansible.cfg configuration file to prevent password authentication:

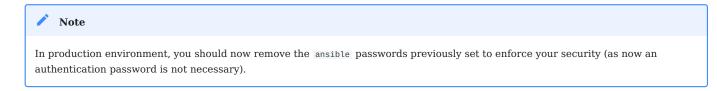
#ask_pass = True

3.5.2 Private key authentication test

For the next test, the shell module, allowing remote command execution, is used:

```
# ansible rocky8 -m shell -a "uptime"
172.16.1.10 | SUCCESS | rc=0 >>
12:36:18 up 57 min, 1 user, load average: 0.00, 0.00, 0.00
172.16.1.11 | SUCCESS | rc=0 >>
12:37:07 up 57 min, 1 user, load average: 0.00, 0.00, 0.00
```

No password is required, private/public key authentication works!



3.6 Using Ansible

Ansible can be used from the shell or via playbooks.

3.6.1 The modules

The list of modules classified by category can be found here. Ansible offers more than 750!

The modules are now grouped into module collections, a list of which can be found here.

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.

A module is invoked with the -m option of the ansible command:

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

There is a module for almost every need! It is thus advised, instead of using the shell module, to look for a module adapted to the need.

Each category of need has its own module. Here is a non-exhaustive list:

Туре	Examples
System Management	user (users management), group (groups management), etc.
Software management	dnf,yum, apt, pip, npm
File management	copy, fetch, lineinfile, template, archive
Database management	mysql, postgresql, redis
Cloud management	amazon S3, cloudstack, openstack
Cluster management	consul, zookeeper
Send commands	shell, script, expect
Downloads	get_url
Source management	git, gitlab

Example of software installation

The dnf module allows for the installation of software on the target clients:

```
# ansible rocky8 --become -m dnf -a name="httpd"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        ...
        \n\nComplete!\n"
    ]
}
```

```
172.16.1.11 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        ...
    \n\nComplete!\n"
    ]
}
```

The installed software being a service, it is now necessary to start it with the module <code>systemd</code> :

```
# ansible rocky8 --become -m systemd -a "name=httpd state=started"
172.16.1.10 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
172.16.1.11 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
```

🜢 Tip

Try to launch those last 2 commands twice. You will observe that the first time Ansible will take actions to reach the state set by the command. The second time, it will do nothing because it will have detected that the state is already reached!

3.6.2 Exercises

To help discover more about Ansible and to get used to searching the Ansible documentation, here are some exercises you can do before going on:

- Create the groups Paris, Tokio, NewYork
- Create the user supervisor
- Change the user to have a uid of 10000
- Change the user so that it belongs to the Paris group
- Install the tree software
- Stop the crond service
- Create an empty file with 644 rights
- Update your client distribution
- Restart your client

Warning
Do not use the shell module. Look in the documentation for the appropriate modules!

setup module: introduction to facts

The system facts are variables retrieved by Ansible via its setup module.

Take a look at the different facts of your clients to get an idea of the amount of information that can be easily retrieved via a simple command.

We'll see later how to use facts in our playbooks and how to create our own facts.

```
# ansible ansible_clients -m setup | less
192.168.1.11 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.1.11"
    ],
    "ansible_all_ipv6_addresses": [
            "2001:861:3dc3:fcf0:a00:27ff:fef7:28be",
            "fe80::a00:27ff:fef7:28be"
    ],
    "ansible_apparmor": {
```

```
"status": "disabled"
},
"ansible_architecture": "x86_64",
"ansible_bios_date": "12/01/2006",
"ansible_bios_vendor": "innotek GmbH",
"ansible_bios_version": "VirtualBox",
"ansible_board_asset_tag": "NA",
"ansible_board_name": "VirtualBox",
"ansible_board_serial": "NA",
"ansible_board_vendor": "Oracle Corporation",
...
```

Now that we have seen how to configure a remote server with Ansible on the command line, we will be able to introduce the notion of playbook. Playbooks are another way to use Ansible, which is not much more complex, but which will make it easier to reuse your code.

3.7 Playbooks

Note

Ansible's playbooks describe a policy to be applied to remote systems, to force their configuration. Playbooks are written in an easily understandable text format that groups together a set of tasks: the yaml format.

```
Learn more about yaml here
```

```
ansible-playbook <file.yml> ... [options]
```

The options are identical to the ansible command.

Code	Error
Θ	OK or no matching host
1	Error
2	One or more hosts are failing
3	One or more hosts are unreachable
4	Analyze error
5	Bad or incomplete options
99	Run interrupted by user
250	Unexpected error

The command returns the following error codes:

Note

Please note that ansible will return Ok when no host matches your target, which might mislead you!

3.7.1 Example of Apache and MySQL playbook

The following playbook allows us to install Apache and MariaDB on our target servers.

Create a test.yml file with the following content:

```
hosts: rocky8 <1>
become: true <2>
become_user: root
tasks:

name: ensure apache is at the latest version
dnf: name=httpd, php, php-mysqli state=latest
name: ensure httpd is started
systemd: name=httpd state=started
name: ensure mariadb is at the latest version
dnf: name=mariadb-server state=latest
name: ensure mariadb is started
```

```
systemd: name=mariadb state=started
...
```

- <1> The targeted group or the targeted server must exist in the inventory
- <2> Once connected, the user becomes root (via sudo by default)

The execution of the playbook is done with the command ansible-playbook :

```
$ ansible-playbook test.yml
ok: [172.16.1.10]
ok: [172.16.1.11]
ok: [172.16.1.10]
ok: [172.16.1.11]
changed: [172.16.1.10]
changed: [172.16.1.11]
TASK [ensure mariadb is at the latest version]
     changed: [172.16.1.10]
changed: [172.16.1.11]
TASK [ensure mariadb is started]
                  * * * * * * * * * * * * * * *
changed: [172.16.1.10]
changed: [172.16.1.11]
PLAY RECAP
: ok=5 changed=3 unreachable=0 failed=0
172.16.1.10
                                    failed=0
172.16.1.11
                   changed=<mark>3</mark>
                          unreachable=0
             : ok=<mark>5</mark>
```

For more readability, it is recommended to write your playbooks in full yaml format. In the previous example, the arguments are given on the same line as the module, the value of the argument following its name separated by an =. Look at the same playbook in full yaml:

```
_ _ _
- hosts: rocky8
  become: true
  become_user: root
  tasks:
     - name: ensure apache is at the latest version
       dnf:
         name: httpd,php,php-mysqli
         state: latest
     - name: ensure httpd is started
       systemd:
         name: httpd
         state: started
     - name: ensure mariadb is at the latest version
       dnf:
         name: mariadb-server
         state: latest
     - name: ensure mariadb is started
       systemd:
         name: mariadb
         state: started
 . . .
🗴 Tip
dnf is one of the modules that allow you to give it a list as argument.
```

Note about collections: Ansible now provides modules in the form of collections. Some modules are provided by default within the ansible.builtin collection, others must be installed manually via the:

```
ansible-galaxy collection install [collectionname]
```

where [collectionname] is the name of the collection (the square brackets here are used to highlight the need to replace this with an actual collection name, and are NOT part of the command).

The previous example should be written like this:

```
- - -
- hosts: rocky8
 become: true
 become_user: root
 tasks:
    - name: ensure apache is at the latest version
     ansible.builtin.dnf:
        name: httpd,php,php-mysqli
        state: latest
    - name: ensure httpd is started
     ansible.builtin.systemd:
        name: httpd
        state: started
    - name: ensure mariadb is at the latest version
     ansible.builtin.dnf:
        name: mariadb-server
        state: latest
   - name: ensure mariadb is started
     ansible.builtin.systemd:
       name: mariadb
        state: started
. . .
```

A playbook is not limited to one target:

```
---
- hosts: webservers
become: true
become_user: root
tasks:
    - name: ensure apache is at the latest version
    ansible.builtin.dnf:
    name: httpd,php,php-mysqli
    state: latest
    - name: ensure httpd is started
    ansible.builtin.systemd:
    name: httpd
    state: started
```

```
hosts: databases
become: true
become_user: root
name: ensure mariadb is at the latest version
ansible.builtin.dnf:
name: mariadb-server
state: latest
name: ensure mariadb is started
ansible.builtin.systemd:
name: mariadb
state: started
```

You can check the syntax of your playbook:

ansible-playbook --syntax-check play.yml

You can also use a "linter" for yaml:

dnf install -y yamllint

then check the yaml syntax of your playbooks:

```
$ yamllint test.yml
test.yml
8:1 error syntax error: could not find expected ':' (syntax)
```

3.8 Exercises results

- Create the groups Paris, Tokio, NewYork
- Create the user supervisor
- Change the user to have a uid of 10000
- Change the user so that it belongs to the Paris group
- Install the tree software
- Stop the crond service
- Create an empty file with 0644 rights

- Update your client distribution
- Restart your client

```
ansible ansible_clients --become -m group -a "name=Paris"
ansible ansible_clients --become -m group -a "name=Tokio"
ansible ansible_clients --become -m group -a "name=NewYork"
ansible ansible_clients --become -m user -a "name=Supervisor"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000"
ansible ansible_clients --become -m user -a "name=Supervisor uid=10000
groups=Paris"
ansible ansible_clients --become -m dnf -a "name=tree"
ansible ansible_clients --become -m systemd -a "name=crond state=stopped"
ansible ansible_clients --become -m copy -a
"content='' dest=/tmp/test force=no mode=0644"
ansible ansible_clients --become -m dnf -a "name=* state=latest"
ansible ansible_clients --become -m dnf -a "name=* state=latest"
```

4. Ansible Intermediate

In this chapter, you will continue to learn how to work with Ansible.

Objectives: In this chapter, you will learn how to:

- ✓ work with variables;
- ✓ use loops;
- ✓ manage state changes and react to them;
- ✓ manage asynchronous tasks.

🕅 ansible, module, playbook

Knowledge: $\star \star \star$ Complexity: $\star \star$

Reading time: 30 minutes

In the previous chapter, you learned how to install Ansible, use it on the command line, and write playbooks to promote the re-usability of your code.

In this chapter, we can start to discover more advanced notions of how to use Ansible and some interesting tasks that you will use regularly.

4.1 The variables

NoteMore information can be found here.

Under Ansible, there are different types of primitive variables:

- strings,
- integers,
- booleans.

These variables can be organized as:

- dictionaries,
- lists.

A variable can be defined in different places, such as a playbook, a role, or the command line.

For example, from a playbook:

```
---
- hosts: apache1
vars:
    port_http: 80
    service:
        debian: apache2
        rhel: httpd
```

or from the command line:

ansible-playbook deploy-http.yml --extra-vars "service=httpd"

Once defined, a variable can be used by calling it between double braces:

- {{ port_http }} for a simple value,
- {{ service['rhel'] }} or {{ service.rhel }} for a dictionary.

For example:

```
- name: make sure apache is started
ansible.builtin.systemd:
    name: "{{ service['rhel'] }}"
    state: started
```

Of course, it is also possible to access the global variables (the **facts**) of Ansible (OS type, IP addresses, VM name, etc.).

4.1.1 Outsourcing variables

Variables can be included in a file external to the playbook, in which case this file must be defined in the playbook with the vars_files directive:

```
---
- hosts: apache1
vars_files:
    - myvariables.yml
```

The myvariables.yml file:

```
---
port_http: 80
ansible.builtin.systemd::
   debian: apache2
   rhel: httpd
```

It can also be added dynamically with the use of the module include_vars:

```
- name: Include secrets.
ansible.builtin.include_vars:
file: vault.yml
```

4.1.2 Display a variable

To display a variable, you have to activate the debug module as follows:

```
- ansible.builtin.debug:
    var: service['debian']
```

You can also use the variable inside a text:

```
- ansible.builtin.debug:
    msg: "Print a variable in a message : {{ service['debian'] }}"
```

4.1.3 Save the return of a task

To save the return of a task and to be able to access it later, you have to use the keyword register inside the task itself.

Use of a stored variable:

```
name: /home content
shell: ls /home
register: homes
name: Print the first directory name
ansible.builtin.debug:
var: homes.stdout_lines[0]
name: Print the first directory name
ansible.builtin.debug:
var: homes.stdout_lines[1]
```

Note

The variable homes.stdout_lines is a list of variables of type string, a way to organize variables we had not yet encountered.

The strings that make up the stored variable can be accessed via the stdout value (which allows you to do things like homes.stdout.find("core") != -1), to exploit them using a loop (see loop), or simply by their indices as seen in the previous example.

4.1.4 Exercises:

- Write a playbook, play-vars.yml, using global variables that print the target's distribution name and major version.
- Write a playbook using the following dictionary to display the services that will be installed:

```
service:
web:
name: apache
rpm: httpd
db:
```

```
name: mariadb
rpm: mariadb-server
```

The default type should be "web".

- Override the type variable using the command line
- Externalize variables in a vars.yml file

4.2 Loop management

A loop allows you to iterate a task over a list, a hash, or a dictionary, for example.

```
Note More information can be found here.
```

A simple example of use, creation of 4 users:

At each iteration of the loop, the value of the list used is stored in the item variable, accessible in the loop code.

Of course, a list can be defined in an external file:

users: - antoine - patrick - steven - xavier

and be used inside the task like this (after having included the vars file):

```
- name: add users
    user:
        name: "{{ item }}"
        state: present
        groups: "users"
        loop: "{{ users }}"
```

We can use the example seen while studying stored variables to improve it. Use of a stored variable:

```
name: /home content
shell: ls /home
register: homes
name: Print the directories name
ansible.builtin.debug:
msg: "Directory => {{ item }}"
loop: "{{ homes.stdout_lines }}"
```

A dictionary can also be used in a loop.

In this case, you must transform the dictionary into an item with a **jinja filter** (jinja is the templating engine used by Ansible): | dict2items.

In the loop, it becomes possible to use item.key, which corresponds to the dictionary key, and item.value, which corresponds to the key's values.

Let's see this through a concrete example, showing the management of the system users:

```
---
- hosts: rocky8
  become: true
  become_user: root
  vars:
    users:
    antoine:
    group: users
    state: present
    steven:
    group: users
    state: absent
tasks:
```

```
- name: Manage users
user:
   name: "{{ item.key }}"
   group: "{{ item.value.group }}"
   state: "{{ item.value.state }}"
loop: "{{ users | dict2items }}"
```

🖍 Note

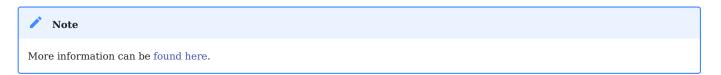
Loops can be used for many things. When your use of Ansible pushes you to use them more complexly, you will discover the possibilities they offer.

4.2.1 Exercises:

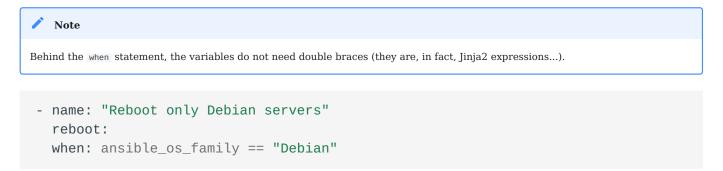
• Display the content of the service variable from the previous exercise using a loop.

```
Note
You will have to transform your service variable, which is a dictionary, to a list with the help of the jinja filter list as this:
{{ service.values() | list }}
```

4.3 Conditionals



The when statement is very useful in many cases, such as not performing certain actions on certain types of servers, if a file or a user does not exist, etc.



Conditions can be grouped with parentheses:

```
- name: "Reboot only CentOS version 6 and Debian version 7"
  reboot:
  when: (ansible_distribution == "CentOS" and
  ansible_distribution_major_version == "6") or
        (ansible_distribution == "Debian" and
  ansible_distribution_major_version == "7")
```

The conditions corresponding to a logical AND can be provided as a list:

```
- name: "Reboot only CentOS version 6"
  reboot:
  when:
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "6"
```

You can test the value of a boolean and verify that it is true:

```
name: check if directory exists
stat:
path: /home/ansible
register: directory
ansible.builtin.debug:
var: directory
ansible.builtin.debug:
msg: The directory exists
when:
- directory.stat.exists
- directory.stat.isdir
```

You can also test that it is not true:

```
when:
- file.stat.exists
- not file.stat.isdir
```

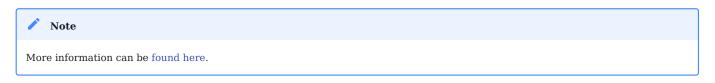
You will probably have to test that a variable exists to avoid execution errors:

when: myboolean is defined and myboolean

4.3.1 Exercises:

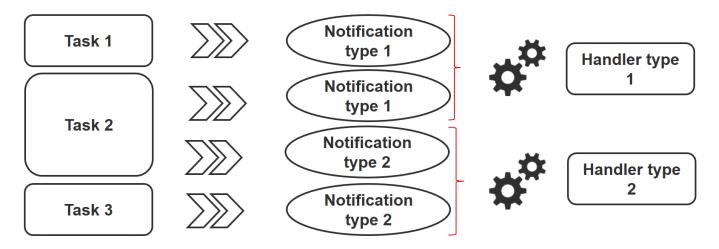
• Print the value of service.web only when type equals to web.

4.4 Managing changes: the handlers



When changes occur, handlers are allowed to launch operations, like restarting a service.

A module, being idempotent, a playbook can detect that there has been a significant change on a remote system and thus trigger an operation in reaction to this change. A notification is sent at the end of a playbook task block, and the reaction operation will be triggered only once, even if several tasks send the same notification.



For example, several tasks may indicate that the httpd service needs to be restarted due to a change in its configuration files. However, the service will only be restarted once to avoid multiple unnecessary starts.

- 39/113 -

```
- name: template configuration file
  template:
    src: template-site.j2
    dest: /etc/httpd/sites-availables/test-site.conf
    notify:
```

restart memcachedrestart httpd

A handler is a kind of task referenced by a unique global name:

- One or more notifiers activate it.
- It does not start immediately, but waits until all tasks are complete to run.

Example of handlers:

```
handlers:
    name: restart memcached
    systemd:
        name: memcached
        state: restarted
        name: restart httpd
        systemd:
        name: httpd
        state: restarted
```

Since version 2.2 of Ansible, handlers can listen directly as well:

```
handlers:
    - name: restart memcached
    systemd:
    name: memcached
    state: restarted
    listen: "web services restart"
    name: restart apache
    systemd:
    name: apache
    state: restarted
    listen: "web services restart"
tasks:
    - name: restart everything
    command: echo "this task will restart the web services"
    notify: "web services restart"
```

4.5 Asynchronous tasks

Note
More information can be found here.

By default, SSH connections to hosts remain open while executing various playbook tasks on all nodes.

This can cause some problems, especially:

- if the execution time of the task is longer than the SSH connection timeout
- if the connection is interrupted during the action (server reboot, for example)

In this case, you will have to switch to asynchronous mode and specify a maximum execution time and the frequency (by default, 10s) with which you will check the host status.

By specifying a poll value of 0, Ansible will execute the task and continue without worrying about the result.

Here's an example using asynchronous tasks, which allows you to restart a server and wait for port 22 to be reachable again:

```
# Wait 2s and launch the reboot
- name: Reboot system
  shell: sleep 2 && shutdown -r now "Ansible reboot triggered"
  async: 1
  poll: 0
  ignore_errors: true
  become: true
  changed_when: False
  # Wait the server is available
  - name: Waiting for server to restart (10 mins max)
    wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
      delay: 30
      state: started
      timeout: 600
    delegate_to: localhost
```

You can also decide to launch a long-running task and forget it (fire and forget) because the execution does not matter in the playbook.

4.6 Exercise results

_ _ _

• Write a playbook, `play-vars.yml, ' using global variables, that prints the target's distribution name and major version.

```
- hosts: ansible_clients
 tasks:
  - name: Print globales variables
   debua:
     msg: "The distribution is {{ ansible_distribution }} version
{{ ansible_distribution_major_version }}"
$ ansible-playbook play-vars.yml
PLAY [ansible_clients]
* *
TASK [Gathering Facts]
* *
ok: [192.168.1.11]
TASK [Print globales variables]
                    *****
ok: [192.168.1.11] => {
  "msg": "The distribution is Rocky version 8"
}
PLAY RECAP
* * * * * * * * * * * * *
192.168.1.11
                : ok=2 changed=0 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0
```

• Write a playbook using the following dictionary to display the services that will be installed:

```
service:
  web:
    name: apache
    rpm: httpd
  db:
    name: mariadb
    rpm: mariadb-server
```

The default type should be "web".

```
- - -
- hosts: ansible_clients
 vars:
   type: web
   service:
    web:
      name: apache
      rpm: httpd
    db:
      name: mariadb
      rpm: mariadb-server
 tasks:
   - name: Print a specific entry of a dictionary
    debug:
      msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
$ ansible-playbook display-dict.yml
PLAY [ansible_clients]
                           * * * * * * * * * * * * * * * * * *
* *
TASK [Gathering Facts]
                   * *
ok: [192.168.1.11]
TASK [Print a specific entry of a dictionnaire]
                     ok: [192.168.1.11] => {
   "msg": "The apache will be installed with the packages httpd"
}
```

• Override the type variable using the command line:

```
ansible-playbook --extra-vars "type=db" display-dict.yml
PLAY [ansible_clients]
* *
TASK [Gathering Facts]
           * *
ok: [192.168.1.11]
TASK [Print a specific entry of a dictionary]
ok: [192.168.1.11] => {
  "msg": "The mariadb will be installed with the packages mariadb-server"
}
PLAY RECAP
* * * * * * * * * * * * * *
                     changed=0 unreachable=0 failed=0
192.168.1.11
               : ok=2
skipped=0 rescued=0 ignored=0
```

• Externalize variables in a vars.yml file

```
type: web
service:
web:
name: apache
rpm: httpd
db:
name: mariadb
rpm: mariadb-server
```

```
- hosts: ansible_clients
vars_files:
```

_ _ _

```
- vars.yml
tasks:
    - name: Print a specific entry of a dictionary
    debug:
    msg: "The {{ service[type]['name'] }} will be installed with the
packages {{ service[type].rpm }}"
```

• Display the content of the service variable from the previous exercise using a loop.

Note You will have to transform your service variable, which is a dictionary, to an item or a list with the help of the jinja filters dict2items or list as this: {{ service | dict2items }}

{{ service.values() | list }}

With dict2items:

```
- hosts: ansible_clients
 vars_files:
   - vars.yml
 tasks:
   - name: Print a dictionary variable with a loop
    debug:
      msg: "{{item.key }} | The {{ item.value.name }} will be installed with
the packages {{ item.value.rpm }}"
     loop: "{{ service | dict2items }}"
$ ansible-playbook display-dict.yml
PLAY [ansible_clients]
                        ++++
* *
TASK [Gathering Facts]
                          ******
* *
ok: [192.168.1.11]
```

```
TASK [Print a dictionary variable with a loop]
* * * * * * * * *
ok: [192.168.1.11] => (item={'key': 'web', 'value': {'name': 'apache', 'rpm':
'httpd'}) => {
   "msg": "web | The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'key': 'db', 'value': {'name': 'mariadb', 'rpm':
'mariadb-server'}}) => {
   "msg": "db | The mariadb will be installed with the packages mariadb-
server"
}
PLAY RECAP
* * * * * * * * * * * * * *
192.168.1.11
                       : ok=2 changed=0 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0
```

With list:

```
- hosts: ansible_clients
 vars_files:
   - vars.yml
  tasks:
    - name: Print a dictionary variable with a loop
      debug:
        msg: "The {{ item.name }} will be installed with the packages
{{ item.rpm }}"
      loop: "{{ service.values() | list}}"
```

```
$ ansible-playbook display-dict.yml
PLAY [ansible_clients]
                   * * * * * * * * * * * * * * *
* *
TASK [Gathering Facts]
                   * *
ok: [192.168.1.11]
```

- 46/113 -

```
TASK [Print a dictionary variable with a loop]
ok: [192.168.1.11] => (item={'name': 'apache', 'rpm': 'httpd'}) => {
   "msg": "The apache will be installed with the packages httpd"
}
ok: [192.168.1.11] => (item={'name': 'mariadb', 'rpm': 'mariadb-server'}) => {
   "msg": "The mariadb will be installed with the packages mariadb-server"
}
PLAY RECAP
* * * * * * * * * * * * *
192.168.1.11
                     : ok=2
                             changed=0 unreachable=0 failed=0
skipped=0 rescued=0
                   ignored=0
```

• Print the value of service.web only when type equals to web.

```
- hosts: ansible_clients
 vars_files:
   - vars.yml
 tasks:
   - name: Print a dictionary variable
     debug:
      msg: "The {{ service.web.name }} will be installed with the packages
{{ service.web.rpm }}"
     when: type == "web"
   - name: Print a dictionary variable
     debug:
       msg: "The {{ service.db.name }} will be installed with the packages
{{ service.db.rpm }}"
     when: type == "db"
$ ansible-playbook display-dict.yml
PLAY [ansible_clients]
* *
TASK [Gathering Facts]
```

```
Copyright \ensuremath{\textcircled{O}} 2023 The Rocky Enterprise Software Foundation
```

* * * * * * * * * * *

* *

```
ok: [192.168.1.11]
TASK [Print a dictionary variable]
ok: [192.168.1.11] => {
  "msg": "The apache will be installed with the packages httpd"
}
TASK [Print a dictionary variable]
    skipping: [192.168.1.11]
PLAY RECAP
* * * * * * * * * * * * * *
192.168.1.11
               : ok=2 changed=0 unreachable=0 failed=0
skipped=1 rescued=0 ignored=0
$ ansible-playbook --extra-vars "type=db" display-dict.yml
PLAY [ansible_clients]
   * *
TASK [Gathering Facts]
* *
ok: [192.168.1.11]
TASK [Print a dictionary variable]
skipping: [192.168.1.11]
TASK [Print a dictionary variable]
             ****
* * * * * * * * * * * * * * * * * * *
ok: [192.168.1.11] => {
  "msg": "The mariadb will be installed with the packages mariadb-server"
}
PLAY RECAP
* * * * * * * * * * * * * *
                      changed=0 unreachable=0 failed=0
192.168.1.11
               : ok=2
skipped=1 rescued=0 ignored=0
```

5. Ansible - Management of Files

In this chapter you will learn how to manage files with Ansible.

Objectives: In this chapter you will learn how to:

modify the content of file;
upload files to the targeted servers;
retrieve files from the targeted servers.

🕅 ansible, module, files

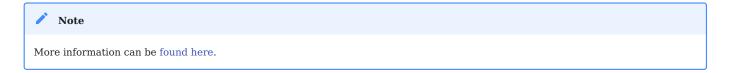
Knowledge: $\star \star$ Complexity: \star

Reading time: 20 minutes

Depending on your needs, you will have to use different Ansible modules to modify the system configuration files.

5.1 ini_file module

When you want to modify an INI file (section between [] then key=value pairs), the easiest way is to use the ini_file module.



The module requires:

- The value of the section
- The name of the option
- The new value

Example of use:

```
    name: change value on inifile
community.general.ini_file:
    dest: /path/to/file.ini
    section: SECTIONNAME
    option: OPTIONNAME
    value: NEWVALUE
```

5.2 lineinfile module

To ensure that a line is present in a file, or when a single line in a file needs to be added or modified, use the linefile module.

```
Note More information can be found here.
```

In this case, the line to be modified in a file will be found using a regexp.

For example, to ensure that the line starting with SELINUX= in the /etc/selinux/ config file contains the value enforcing:

```
- ansible.builtin.lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'
```

5.3 copy module

When a file has to be copied from the Ansible server to one or more hosts, it is better to use the copy module.

```
    Note
More information can be found here.
```

Here we are copying myflile.conf from one location to another:

```
- ansible.builtin.copy:
    src: /data/ansible/sources/myfile.conf
    dest: /etc/myfile.conf
    owner: root
```

group: root mode: 0644

5.4 fetch module

When a file has to be copied from a remote server to the local server, it is best to use the fetch module.

```
Note More information can be found here.
```

This module does the opposite of the copy module:

```
- ansible.builtin.fetch:
    src: /etc/myfile.conf
    dest: /data/ansible/backup/myfile-{{ inventory_hostname }}.conf
    flat: yes
```

5.5 template module

Ansible and its template module use the **Jinja2** template system (http://jinja.pocoo.org/docs/) to generate files on target hosts.

```
/ Note
```

More information can be found here.

For example:

```
- ansible.builtin.template:
    src: /data/ansible/templates/monfichier.j2
    dest: /etc/myfile.conf
    owner: root
    group: root
    mode: 0644
```

It is possible to add a validation step if the targeted service allows it (for example apache with the command apachectl -t):

```
- template:
    src: /data/ansible/templates/vhost.j2
    dest: /etc/httpd/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
    validate: '/usr/sbin/apachectl -t'
```

5.6 get_url module

To upload files from a web site or ftp to one or more hosts, use the get_url module:

```
- get_url:
    url: http://site.com/archive.zip
    dest: /tmp/archive.zip
    mode: 0640
    checksum:
sha256:f772bd36185515581aa9a2e4b38fb97940ff28764900ba708e68286121770e9a
```

By providing a checksum of the file, the file will not be re-downloaded if it is already present at the destination location and its checksum matches the value provided.

6. Ansible Galaxy: Collections and Roles

In this chapter you will learn how to use, install, and manage Ansible roles and collections.

Objectives: In this chapter you will learn how to:

 \checkmark install and manage collections.

 \checkmark install and manage roles.

🕅 ansible, ansible-galaxy, roles, collections

Knowledge: $\star \star$ Complexity: $\star \star \star$

Reading time: 40 minutes

Ansible Galaxy provides Ansible Roles and Collections from the Ansible Community.

The elements provided can be referenced in the playbooks and used out of the box

6.1 ansible-galaxy command

The ansible-galaxy command manages roles and collections using galaxy.ansible.com.

• To manage roles:

ansible-galaxy role [import|init|install|login|remove|...]

Sub-commands	Functionality
install	installs a role.
remove	remove one or more roles.
list	display the name and the version of installed roles.
info	display information about a role.
init	generate a skeleton of a new role.
import	import a role from the galaxy web site. Requires a login.

• To manage collections:

ansible-galaxy collection [import|init|install|login|remove|...]

Sub-commands	Functionality
init	generate a skeleton of a new collection.
install	installs a collection.
list	display the name and the version of installed collections.

6.2 Ansible Roles

An Ansible role is a unit that promotes the reusability of playbooks.



6.2.1 Installing useful Roles

In order to highlight the interest of using roles, I suggest you to use the alemorvan/ patchmanagement role, which will allow you to perform a lot of tasks (pre-update or post-update for example) during your update process, in only a few lines of code.

You can check the code in the github repo of the role here.

• Install the role. This needs only one command:

ansible-galaxy role install alemorvan.patchmanagement

• Create a playbook to include the role:

```
- name: Start a Patch Management
hosts: ansible_clients
vars:
    pm_before_update_tasks_file: custom_tasks/pm_before_update_tasks_file.yml
    pm_after_update_tasks_file: custom_tasks/pm_after_update_tasks_file.yml
    tasks:
        - name: "Include patchmanagement"
        include_role:
            name: "alemorvan.patchmanagement"
```

With this role, you can add your own tasks for all your inventory or for only your targeted node.

Let's create tasks that will be run before and after the update process:

• Create the custom_tasks folder:

mkdir custom_tasks

• Create the custom_tasks/pm_before_update_tasks_file.yml (feel free to change the name and the content of this file)

```
    name: sample task before the update process
debug:
msg: "This is a sample tasks, feel free to add your own test task"
```

• Create the custom_tasks/pm_after_update_tasks_file.yml (feel free to change the name and the content of this file)

```
- name: sample task after the update process
  debug:
    msg: "This is a sample tasks, feel free to add your own test task"
```

And launch your first Patch Management:

```
ansible-playbook patchmanagement.yml
PLAY [Start a Patch Management]
TASK [Gathering Facts]
* * *
ok: [192.168.1.11]
TASK [Include patchmanagement]
TASK [alemorvan.patchmanagement : MAIN | Linux Patch Management Job]
ok: [192.168.1.11] => {
  "msq": "Start 192 patch management"
}
. . .
TASK [alemorvan.patchmanagement : sample task before the update process]
ok: [192.168.1.11] => {
  "msg": "This is a sample tasks, feel free to add your own test task"
}
. . .
TASK [alemorvan.patchmanagement : MAIN | We can now patch]
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
patch.yml for 192.168.1.11
TASK [alemorvan.patchmanagement : PATCH | Tasks depends on distribution]
ok: [192.168.1.11] => {
  "ansible_distribution": "Rocky"
}
TASK [alemorvan.patchmanagement : PATCH | Include tasks for CentOS & RedHat
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
linux_tasks/redhat_centos.yml for 192.168.1.11
```

TASK [alemorvan.patchmanagement : RHEL CENTOS | yum clean all]

```
changed: [192.168.1.11]
TASK [alemorvan.patchmanagement : RHEL CENTOS | Ensure yum-utils is installed]
ok: [192.168.1.11]
TASK [alemorvan.patchmanagement : RHEL CENTOS | Remove old kernels]
skipping: [192.168.1.11]
TASK [alemorvan.patchmanagement : RHEL CENTOS | Update rpm package with yum]
ok: [192.168.1.11]
TASK [alemorvan.patchmanagement : PATCH | Inlude tasks for Debian & Ubuntu
skipping: [192.168.1.11]
TASK [alemorvan.patchmanagement : MAIN | We can now reboot]
included: /home/ansible/.ansible/roles/alemorvan.patchmanagement/tasks/
reboot.yml for 192.168.1.11
TASK [alemorvan.patchmanagement : REBOOT | Reboot triggered]
ok: [192.168.1.11]
TASK [alemorvan.patchmanagement : REBOOT | Ensure we are not in rescue mode]
ok: [192.168.1.11]
. . .
TASK [alemorvan.patchmanagement : FACTS | Insert fact file]
ok: [192.168.1.11]
TASK [alemorvan.patchmanagement : FACTS | Save date of last PM]
ok: [192.168.1.11]
. . .
TASK [alemorvan.patchmanagement : sample task after the update process]
ok: [192.168.1.11] => {
   "msg": "This is a sample tasks, feel free to add your own test task"
}
```

PLAY RECAP ************************************								
* * * * * * * * * * * * *								
<mark>192</mark> .168.1.11		: ok= 31	changed=1	unreachable=0	failed=0			
skipped= <mark>4</mark>	rescued=0	ignored= <mark>0</mark>						

Pretty easy for such a complex process, isn't it?

This is just one example of what can be done using roles made available by the community. Have a look at galaxy.ansible.com to discover the roles that could be useful for you!

You can also create your own roles for your own needs and publish them on the Internet if you feel like it. This is what we will briefly cover in the next chapter.

6.2.2 Introduction to Role development

A role skeleton, serving as a starting point for custom role development, can be generated by the ansible-galaxy command:

```
$ ansible-galaxy role init rocky8
```

- Role rocky8 was created successfully

The command will generate the following tree structure to contain the rocky8 role:

```
tree rocky8/
rocky8/
├── 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
```

Roles allow you to do away with the need to include files. There is no need to specify file paths or include directives in playbooks. You just have to specify a task, and Ansible takes care of the inclusions.

The structure of a role is fairly obvious to understand.

Variables are simply stored either in vars/main.yml if the variables are not to be overridden, or in default/main.yml if you want to leave the possibility of overriding the variable content from outside your role.

The handlers, files, and templates needed for your code are stored in handlers/ main.yml, files and templates respectively.

All that remains is to define the code for your role's tasks in tasks/main.yml.

Once all this is working well, you can use this role in your playbooks. You will be able to use your role without worrying about the technical aspect of its tasks, while customizing its operation with variables.

6.2.3 Practical work: create a first simple role

Let's implement this with a "go anywhere" role that will create a default user and install software packages. This role can be systematically applied to all your servers.

Variables

We will create a rockstar user on all of our servers. As we don't want this user to be overridden, let's define it in the vars/main.yml:

```
rocky8_default_group:
    name: rockstar
    gid: 1100
rocky8_default_user:
    name: rockstar
```

uid: <mark>1100</mark> group: rockstar

We can now use those variables inside our tasks/main.yml without any inclusion.

```
- name: Create default group
group:
name: "{{ rocky8_default_group.name }}"
gid: "{{ rocky8_default_group.gid }}"
- name: Create default user
user:
name: "{{ rocky8_default_user.name }}"
uid: "{{ rocky8_default_user.uid }}"
group: "{{ rocky8_default_user.group }}"
```

To test your new role, let's create a test-role.yml playbook in the same directory as your role:

```
---
- name: Test my role
hosts: localhost
roles:
    - role: rocky8
    become: true
    become_user: root
```

and launch it:

```
ansible-playbook test-role.yml
PLAY [Test my role]
******
TASK [Gathering Facts]
***
ok: [localhost]
TASK [rocky8 : Create default group]
```

Congratulations! You are now able to create great things with a playbook of only a few lines.

Let's see the use of default variables.

Create a list of packages to install by default on your servers and an empty list of packages to uninstall. Edit the defaults/main.yml files and add those two lists:

```
rocky8_default_packages:
    - tree
    - vim
rocky8_remove_packages: []
```

and use them in your tasks/main.yml:

```
- name: Install default packages (can be overridden)
package:
    name: "{{ rocky8_default_packages }}"
    state: present
- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
    package:
    name: "{{ rocky8_remove_packages }}"
    state: absent
```

Test your role with the help of the playbook previously created:

```
* * * * *
TASK [Gathering Facts]
                * *
ok: [localhost]
TASK [rocky8 : Create default group]
       ok: [localhost]
TASK [rocky8 : Create default user]
       ok: [localhost]
TASK [rocky8 : Install default packages (can be overridden)]
        ok: [localhost]
TASK [rocky8 : Uninstall default packages (can be overridden) []]
ok: [localhost]
PLAY RECAP
* * * * * * * * * * * * * *
               : ok=5 changed=0 unreachable=0 failed=0
localhost
skipped=0 rescued=0 ignored=0
```

You can now override the rocky8_remove_packages in your playbook and uninstall for example cockpit :

```
---
- name: Test my role
hosts: localhost
vars:
    rocky8_remove_packages:
        - cockpit
roles:
        - role: rocky8
        become: true
        become_user: root
```

```
ansible-playbook test-role.yml
PLAY [Test my role]
            * * * * *
TASK [Gathering Facts]
                * * * * * * * * * * * * * * * * *
* *
ok: [localhost]
TASK [rocky8 : Create default group]
                       ok: [localhost]
TASK [rocky8 : Create default user]
                        ok: [localhost]
TASK [rocky8 : Install default packages (can be overridden)]
         ok: [localhost]
TASK [rocky8 : Uninstall default packages (can be overridden) ['cockpit']]
      changed: [localhost]
PLAY RECAP
* * * * * * * * * * * * * *
localhost
                 : ok=5 changed=1 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0
```

Obviously, there is no limit to how much you can improve your role. Imagine that for one of your servers, you need a package that is in the list of those to be uninstalled. You could then, for example, create a new list that can be overridden and then remove from the list of packages to be uninstalled those in the list of specific packages to be installed by using the jinja difference() filter.

```
- name: "Uninstall default packages (can be overridden)
{{ rocky8_remove_packages }}"
   package:
    name: "{{ rocky8_remove_packages |
   difference(rocky8_specifics_packages) }}"
    state: absent
```

6.3 Ansible Collections

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.

```
        Note

        More information can be found here
```

To install or upgrade a collection:

ansible-galaxy collection install namespace.collection [--upgrade]

You can then use the newly installed collection using its namespace and name before the module's name or role's name:

```
- import_role:
    name: namespace.collection.rolename
```

- namespace.collection.modulename:
 option1: value

You can find a collection index here.

Let's install the community.general collection:

```
ansible-galaxy collection install community.general
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/community-general-3.3.2.tar.gz
to /home/ansible/.ansible/tmp/ansible-local-51384hsuhf3t5/tmpr_c9qrt1/
community-general-3.3.2-f4q9u4dg
Installing 'community.general:3.3.2' to '/home/ansible/.ansible/collections/
ansible_collections/community/general'
community.general:3.3.2 was installed successfully
```

We can now use the newly available module yum_versionlock :

```
- name: Start a Patch Management
hosts: ansible_clients
become: true
become_user: root
```

```
tasks:
  - name: Ensure yum-versionlock is installed
   package:
     name: python3-dnf-plugin-versionlock
     state: present
  - name: Prevent kernel from being updated
   community.general.yum_versionlock:
     state: present
     name: kernel
   register: locks
  - name: Display locks
   debug:
     var: locks.meta.packages
ansible-playbook versionlock.yml
PLAY [Start a Patch Management]
                      TASK [Gathering Facts]
* * *
ok: [192.168.1.11]
TASK [Ensure yum-versionlock is installed]
changed: [192.168.1.11]
TASK [Prevent kernel from being updated]
changed: [192.168.1.11]
TASK [Display locks]
                * * * * *
ok: [192.168.1.11] => {
  "locks.meta.packages": [
     "kernel"
  1
}
PLAY RECAP
* * * * * * * * * * * * * * *
```

<mark>192</mark> .168.1.11	L	: ok=4	changed= <mark>2</mark>	unreachable= <mark>0</mark>	failed=0
skipped=0	rescued=0	ignored= <mark>0</mark>			

6.3.1 Creating your own collection

As with roles, you are able to create your own collection with the help of the ansible-galaxy command:

You can then store your own plugins or roles inside this new collection.

7. Ansible Deployments with Ansistrano

In this chapter you will learn how to deploy applications with the Ansible role Ansistrano.

Objectives: In this chapter you will learn how to:

- ✓ Implement Ansistrano;
- ✓ Configure Ansistrano;
- ✓ Use shared folders and files between deployed versions;
- Deploying different versions of a site from git;
- \checkmark React between deployment steps.

🕅 ansible, ansistrano, roles, deployments

Knowledge: $\star \star$ Complexity: $\star \star \star$

Reading time: 40 minutes

Ansistrano is an Ansible role to easily deploy PHP, Python, etc. applications. It is based on the functionality of Capistrano.

7.1 Introduction

Ansistrano requires the following to run:

- Ansible on the deployment machine,
- rsync or git on the client machine.

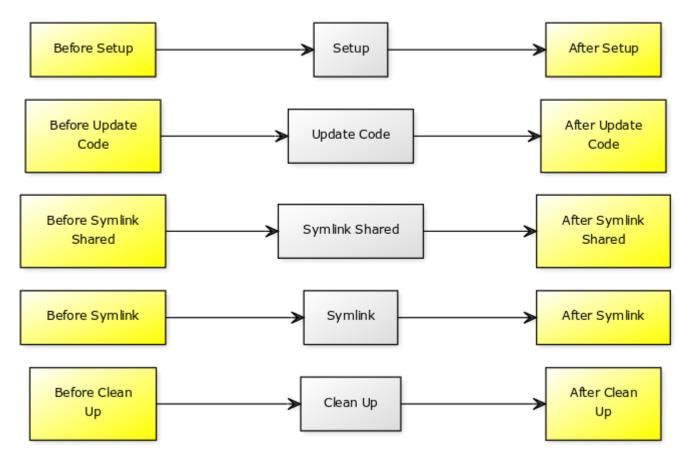
It can download source code from <code>rsync</code>, <code>git</code>, <code>scp</code>, <code>http</code>, <code>S3</code>, ...

🖍 Note

For our deployment example, we will use the $\ensuremath{\,\mbox{git}}$ protocol.

Ansistrano deploys applications by following these 5 steps:

- **Setup**: create the directory structure to host the releases;
- Update Code: downloading the new release to the targets;
- **Symlink Shared** and **Symlink**: after deploying the new release, the current symbolic link is modified to point to this new release;
- Clean Up: to do some clean up (remove old versions).



The skeleton of a deployment with Ansistrano looks like this:



- 68/113 -

You can find all the Ansistrano documentation on its Github repository.

7.2 Labs

You will continue to work on your 2 servers:

The management server:

• Ansible is already installed. You will have to install the ansistrano.deploy role.

The managed server:

• You will need to install Apache and deploy the client site.

7.2.1 Deploying the Web server

For more efficiency, we will use the geerlingguy.apache role to configure the server:

```
$ ansible-galaxy role install geerlingguy.apache
Starting galaxy role install process
- downloading role 'apache', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-apache/
archive/3.1.4.tar.gz
- extracting geerlingguy.apache to /home/ansible/.ansible/roles/
geerlingguy.apache
- geerlingguy.apache (3.1.4) was installed successfully
```

We will probably need to open some firewall rules, so we will also install the collection ansible.posix to work with its module firewalld:

```
$ ansible-galaxy collection install ansible.posix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ansible-posix-1.2.0.tar.gz to /
home/ansible/.ansible/tmp/ansible-local-519039bp65pwn/tmpsvuj1fw5/ansible-
posix-1.2.0-bhjbfdpw
Installing 'ansible.posix:1.2.0' to '/home/ansible/.ansible/collections/
ansible_collections/ansible/posix'
ansible.posix:1.2.0 was installed successfully
```

Once the role and the collection are installed, we can create the first part of our playbook, which will:

- Install Apache,
- Create a target folder for our vhost ,
- Create a default vhost,
- Open the firewall,
- Start or restart Apache.

Technical considerations:

- We will deploy our site to the /var/www/site/ folder.
- As we will see later, ansistrano will create a current symbolic link to the current release folder.
- The source code to be deployed contains a html folder which the vhost should point to. Its DirectoryIndex is index.htm.
- The deployment is done by git, the package will be installed.

```
Note
The target of our vhost will therefore be: /var/www/site/current/html.
```

Our playbook to configure the server: playbook-config-server.yml

```
----
- hosts: ansible_clients
become: yes
become_user: root
vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
    DirectoryIndex index.php index.htm
    apache_vhosts:
        - servername: "website"
    documentroot: "{{ dest }}current/html"
    tasks:
        - name: create directory for website
        file:
```

```
path: /var/www/site/
state: directory
mode: 0755

- name: install git
package:
    name: git
    state: latest

- name: permit traffic in default zone for http service
ansible.posix.firewalld:
    service: http
    permanent: yes
    state: enabled
    immediate: yes

roles:
    - { role: geerlingguy.apache }
```

The playbook can be applied to the server:

ansible-playbook playbook-config-server.yml

Note the execution of the following tasks:

The geerlingguy.apache role makes our job much easier by taking care of the installation and configuration of Apache.

You can check that everything is working by using curl:

```
$ curl -I http://192.168.1.11
HTTP/1.1 404 Not Found
```

```
Date: Mon, 05 Jul 2021 23:30:02 GMT
Server: Apache/2.4.37 (rocky) OpenSSL/1.1.1g
Content-Type: text/html; charset=iso-8859-1
```

Note

We have not yet deployed any code, so it is normal for curl to return a 404 HTTP code. But we can already confirm that the httpd service is working and that the firewall is open.

7.2.2 Deploying the software

Now that our server is configured, we can deploy the application.

For this, we will use the ansistrano.deploy role in a second playbook dedicated to application deployment (for more readability).

```
$ ansible-galaxy role install ansistrano.deploy
Starting galaxy role install process
- downloading role 'deploy', owned by ansistrano
- downloading role from https://github.com/ansistrano/deploy/archive/
3.10.0.tar.gz
- extracting ansistrano.deploy to /home/ansible/.ansible/roles/
ansistrano.deploy
- ansistrano.deploy (3.10.0) was installed successfully
```

The sources of the software can be found in the github repository.

We will create a playbook playbook-deploy.yml to manage our deployment:

```
---
- hosts: ansible_clients
become: yes
become_user: root
vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
roles:
    - { role: ansistrano.deploy }
```

```
$ ansible-playbook playbook-deploy.yml
PLAY [ansible_clients]
TASK [ansistrano.deploy : ANSISTRANO | Ensure deployment base path exists]
* * * * *
TASK [ansistrano.deploy : ANSISTRANO | Ensure releases folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared elements folder exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths exists]
TASK [ansistrano.deploy : ANSISTRANO | Ensure basedir shared files exists]
TASK [ansistrano.deploy : ANSISTRANO | Get release version]
* * * * * * * * * * * * *
TASK [ansistrano.deploy : ANSISTRANO | Get release path]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Register ansistrano_git_result
variable]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Set git_real_repo_tree]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Create release folder]
TASK [ansistrano.deploy : ANSISTRANO | GIT | Sync repo subtree[""] to release
path]
TASK [ansistrano.deploy : ANSISTRANO | Copy git released version into REVISION
file]
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and
files]
TASK [ansistrano.deploy : ANSISTRANO | Ensure .rsync-filter is absent]
TASK [ansistrano.deploy : ANSISTRANO | Setup .rsync-filter with shared-folders]
TASK [ansistrano.deploy : ANSISTRANO | Get current folder]
TASK [ansistrano.deploy : ANSISTRANO | Remove current folder if it's a
directory]
TASK [ansistrano.deploy : ANSISTRANO | Change softlink to new release]
TASK [ansistrano.deploy : ANSISTRANO | Clean up releases]
PLAY RECAP
192.168.1.11 : ok=25
                     changed=<mark>8</mark>
                                unreachable=0 failed=0
                                                           skipped=14
rescued=0
           ignored=0
```

So many things done with only 11 lines of code!

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
```

```
<body>
<h1>Version Master</h1>
</body>
<html>
```

7.2.3 Checking on the server

You can now connect by ssh to your client machine.

• Make a tree on the /var/www/site/ directory:

Please note:

- the current symlink to the release ./releases/20210722155312Z
- the presence of a directory shared
- the presence of the git repos in ./repo/
- From the Ansible server, restart the deployment 3 times, then check on the client.

Please note:

- ansistrano kept the 4 last releases,
- the current link linked now to the lastest release

7.2.4 Limit the number of releases

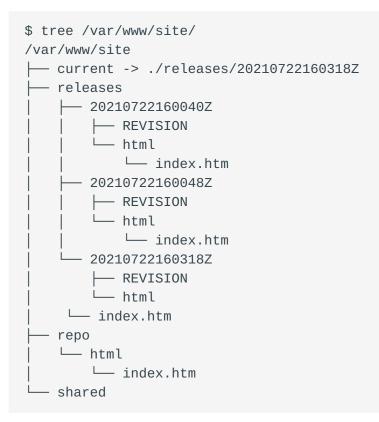
The ansistrano_keep_releases variable is used to specify the number of releases to keep.

• Using the ansistrano_keep_releases variable, keep only 3 releases of the project. Check.

```
----
- hosts: ansible_clients
become: yes
become_user: root
vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
   ansistrano_keep_releases: 3
roles:
   - { role: ansistrano.deploy }
```

\$ ansible-playbook -i hosts playbook-deploy.yml

On the client machine:



7.2.5 Using shared_paths and shared_files

```
_ _ _
- hosts: ansible_clients
 become: yes
 become_user: root
 vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
   ansistrano_keep_releases: 3
   ansistrano_shared_paths:
     - "img"
      - "css"
   ansistrano_shared_files:
      - "logs"
 roles:
    - { role: ansistrano.deploy }
```

On the client machine, create the file logs in the shared directory:

sudo touch /var/www/site/shared/logs

Then execute the playbook:

On the client machine:

```
$ tree -F /var/www/site/
/var/www/site/
— current -> ./releases/20210722160631Z/
  - releases/
     — 20210722160048Z/
        - REVISION
        L
          - html/
            └── index.htm
       - 20210722160318Z/
        - REVISION
          - html/
            └── index.htm
      - 20210722160631Z/
        ├── REVISION
          — css -> ../../shared/css/
         — html/
           └── index.htm
          - img -> ../../shared/img/
        └── logs -> ../../shared/logs
  - repo/
    └── html/
        └── index.htm
  - shared/
    ├─ css/
```

- 77/113 -

– img∕ – logs

Please note that the last release contains 3 links: css, img, and logs

- from /var/www/site/releases/css to the ../../shared/css/ directory.
- from /var/www/site/releases/img to the ../../shared/img/ directory.
- from /var/www/site/releases/logs to the ../../shared/logs file.

Therefore, the files contained in these 2 folders and the logs file are always accessible via the following paths:

- /var/www/site/current/css/,
- /var/www/site/current/img/,
- /var/www/site/current/logs,

but above all they will be kept from one release to the next.

7.2.6 Use a sub-directory of the repository for deployment

In our case, the repository contains a html folder, which contains the site files.

• To avoid this extra level of directory, use the ansistrano_git_repo_tree variable by specifying the path of the sub-directory to use.

Don't forget to modify the Apache configuration to take into account this change!

Change the playbook for the server configuration playbook-config-server.yml

```
- hosts: ansible_clients
  become: yes
  become_user: root
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
    DirectoryIndex index.php index.htm
    apache_vhosts:
        - servername: "website"
  documentroot: "{{ dest }}current/" # <1>
```

```
tasks:
        - name: create directory for website
        file:
path: /var/www/site/
state: directory
mode: 0755
        - name: install git
        package:
name: git
state: latest
roles:
        - { role: geerlingguy.apache }
```

<1> Modify this line

Change the playbook for the deployment playbook-deploy.yml

```
- hosts: ansible_clients
 become: yes
 become_user: root
 vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
   ansistrano_keep_releases: 3
   ansistrano_shared_paths:
      - "img"
      - "CSS"
   ansistrano_shared_files:
      - "log"
   ansistrano_git_repo_tree: 'html' # <1>
 roles:
    - { role: ansistrano.deploy }
```

<1> Modify this line

- Don't forget to run both of the playbooks
- Check on the client machine:

```
$ tree -F /var/www/site/
/var/www/site/
 — current -> ./releases/20210722161542Z/
 — releases/
    ← 20210722160318Z/
        - REVISION
        └── html/
            └── index.htm
      - 20210722160631Z/
        - REVISION
         — css -> ../../shared/css/
         — html/
         └── index.htm
          - img -> ../../shared/img/
        └── logs -> ../../shared/logs
      - 20210722161542Z/
        - REVISION
        └── css -> ../../shared/css/
        └── img -> ../../shared/img/
         — index.htm
         — logs -> ../../shared/logs
  - repo/
    └── html/
       └── index.htm
  - shared/
    ├─ css/
    ⊣ img/
    └── logs
```

<1> Please note the absence of html

7.2.7 Managing git branch or tags

The ansistrano_git_branch variable is used to specify a branch or tag to deploy.

• Deploy the releases/v1.1.0 branch:

```
---
- hosts: ansible_clients
   become: yes
   become_user: root
   vars:
     dest: "/var/www/site/"
     ansistrano_deploy_via: "git"
     ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
```

```
ansistrano_deploy_to: "{{ dest }}"
ansistrano_keep_releases: 3
ansistrano_shared_paths:
    "img"
    "css"
ansistrano_shared_files:
    "log"
ansistrano_git_repo_tree: 'html'
ansistrano_git_branch: 'releases/v1.1.0'
roles:
    { role: ansistrano.deploy }
```

Note

You can have fun, during the deployment, refreshing your browser, to see in 'live' the change.

```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 1.0.1</h1>
</body>
<html>
```

• Deploy the v2.0.0 tag:

```
_ _ _
- hosts: ansible_clients
 become: yes
 become_user: root
 vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
   ansistrano_keep_releases: 3
   ansistrano_shared_paths:
      - "img"
      - "CSS"
   ansistrano_shared_files:
      - "log"
   ansistrano_git_repo_tree: 'html'
   ansistrano_git_branch: 'v2.0.0'
```

```
roles:
    - { role: ansistrano.deploy }
```

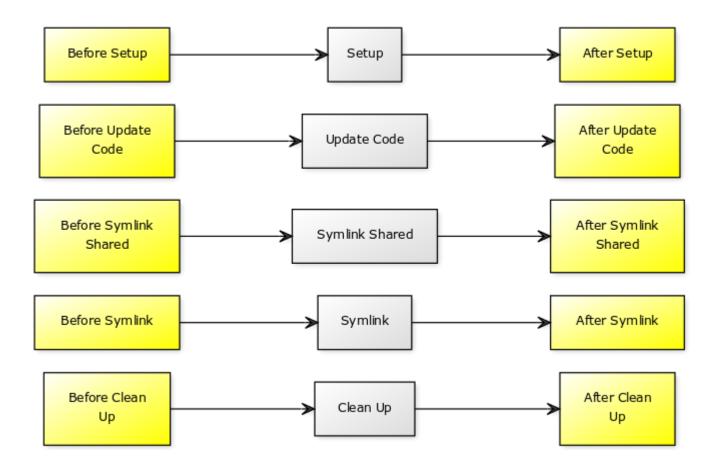
```
$ curl http://192.168.1.11
<html>
<head>
<title>Demo Ansible</title>
</head>
<body>
<h1>Version 2.0.0</h1>
</body>
<html>
```

7.2.8 Actions between deployment steps

A deployment with Ansistrano respects the following steps:

- Setup
- Update Code
- Symlink Shared
- Symlink
- Clean Up

It is possible to intervene before and after each of these steps.



A playbook can be included through the variables provided for this purpose:

- ansistrano_before_<task>_tasks_file
- Or ansistrano_after_<task>_tasks_file
- Easy example: send an email (or whatever you want like Slack notification) at the beginning of the deployment:

```
---
- hosts: ansible_clients
become: yes
become_user: root
vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
        - "img"
        - "css"
    ansistrano_shared_files:
        - "logs"
```

```
ansistrano_git_repo_tree: 'html'
ansistrano_git_branch: 'v2.0.0'
ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"
roles:
    - { role: ansistrano.deploy }
```

Create the file deploy/before-setup-tasks.yml:

```
- name: Send a mail
mail:
    subject: Starting deployment on {{ ansible_hostname }}.
    delegate_to: localhost
TASK [ansistrano.deploy : include]
*******
included: /home/ansible/deploy/before-setup-tasks.yml for 192.168.10.11
TASK [ansistrano.deploy : Send a mail]
*******
ok: [192.168.10.11 -> localhost]
```

[root] # mailx
Heirloom Mail version 12.5 7/5/10. Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N 1 root@localhost.local Tue Aug 21 14:41 28/946 "Starting deployment on
localhost."

• You will probably have to restart some services at the end of the deployment, to flush caches for example. Let's restart Apache at the end of the deployment:

```
---
- hosts: ansible_clients
become: yes
become_user: root
vars:
   dest: "/var/www/site/"
   ansistrano_deploy_via: "git"
   ansistrano_deploy_via: "git"
   ansistrano_git_repo: https://github.com/alemorvan/demo-ansible.git
   ansistrano_deploy_to: "{{ dest }}"
```

```
ansistrano_keep_releases: 3
ansistrano_shared_paths:
    "img"
    "css"
ansistrano_shared_files:
    "logs"
ansistrano_git_repo_tree: 'html'
ansistrano_git_branch: 'v2.0.0'
ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-
setup-tasks.yml"
    ansistrano_after_symlink_tasks_file: "{{ playbook_dir }}/deploy/after-
symlink-tasks.yml"
    roles:
        - { role: ansistrano.deploy }
```

Create the file deploy/after-symlink-tasks.yml:

```
---
- name: restart apache
systemd:
    name: httpd
    state: restarted

TASK [ansistrano.deploy : include]
*******
included: /home/ansible/deploy/after-symlink-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : restart apache]
*******
changed: [192.168.10.11]
```

As you have seen during this chapter, Ansible can greatly improve the life of the system administrator. Very intelligent roles like Ansistrano are "must haves" that quickly become indispensable.

Using Ansistrano, ensures that good deployment practices are respected, reduces the time needed to put a system into production, and avoids the risk of potential human errors. The machine works fast, well, and rarely makes mistakes!

8. Ansible - Large Scale infrastructure

In this chapter you will learn how to scale your configuration management system.

Objectives: In this chapter you will learn how to:

 \checkmark Organize your code for large infrastructure;

 \checkmark Apply all or part of your configuration management to a group of nodes;

🕅 ansible, config management, scale

Knowledge: $\star \star \star$ Complexity: $\star \star \star$

Reading time: 30 minutes

We have seen in the previous chapters how to organize our code in the form of roles but also how to use some roles for the management of updates (patch management) or the deployment of code.

What about configuration management? How to manage the configuration of tens, hundreds, or even thousands of virtual machines with Ansible?

The advent of the cloud has changed the traditional methods a bit. The VM is configured at deployment. If its configuration is no longer compliant, it is destroyed and replaced by a new one.

The organization of the configuration management system presented in this chapter will respond to these two ways of consuming IT: "one-shot" use or regular "re-configuration" of a pool of servers.

However, be careful: using Ansible to ensure a pool of servers compliance requires changing work habits. It is no longer possible to manually modify the configuration of a service manager without seeing these modifications overwritten the next time Ansible is run.

✓ Note
What we are going to set up below is not Ansible's favorite terrain. Technologies like Puppet or Salt will do much better. Let's remember that Ansible is a Swiss army knife of automation and is agentless, which explains the differences in performance.
Note
More information can be found here

8.1 Variables storage

The first thing we have to discuss is the separation between data and Ansible code.

As the code gets larger and more complex, it will be more and more complicated to modify the variables it contains.

To ensure the maintenance of your site, the most important thing is correctly separating the variables from the Ansible code.

We haven't discussed it here yet, but you should know that Ansible can automatically load the variables it finds in specific folders depending on the inventory name of the managed node, or its member groups.

The Ansible documentation suggests that we organize our code as below:

```
inventories/
production/
hosts  # inventory file for production servers
group_vars/
group1.yml  # here we assign variables to particular groups
group2.yml
host_vars/
hostname1.yml  # here we assign variables to particular systems
hostname2.yml
```

If the targeted node is hostname1 of group1, the variables contained in the hostname1.yml and group1.yml files will be automatically loaded. It's a nice way to store all the data for all your roles in the same place.

In this way, the inventory file of your server becomes its identity card. It contains all the variables that differ from the default variables for your server. From the point of view of centralization of variables, it becomes essential to organize the naming of its variables in the roles by prefixing them, for example, with the name of the role. It is also recommended to use flat variable names rather than dictionaries.

For example, if you want to make the PermitRootLogin value in the sshd_config file a variable, a good variable name could be sshd_config_permitrootlogin (instead of sshd.config.permitrootlogin which could also be a good variable name).

8.2 About Ansible tags

The use of Ansible tags allows you to execute or skip a part of the tasks in your code.

```
Note More information can be found here
```

For example, let's modify our users creation task:

You can now play only the tasks with the tag users with the ansible-playbook option --tags:

```
ansible-playbook -i inventories/production/hosts --tags users site.yml
```

You can also use the --skip-tags option.

8.3 About the directory layout

Let's focus on a proposal for the organization of files and directories necessary for the proper functioning of a CMS (Content Management System).

Our starting point will be the site.yml file. This file is a bit like the orchestra conductor of the CMS since it will only include the necessary roles for the target nodes if needed:

```
---
- name: "Config Management for {{ target }}"
hosts: "{{ target }}"
roles:
    - role: roles/functionality1
    - role: roles/functionality2
```

Of course, those roles must be created under the roles directory at the same level as the site.yml file.

I like to manage my global vars inside a vars/global_vars.yml, even if I could store them inside a file located at inventories/production/group_vars/all.yml

```
---
- name: "Config Management for {{ target }}"
hosts: "{{ target }}"
vars_files:
    - vars/global_vars.yml
roles:
    - role: roles/functionality1
    - role: roles/functionality2
```

I also like to keep the possibility of disabling a functionality. So I include my roles with a condition and a default value like this:

```
- name: "Config Management for {{ target }}"
hosts: "{{ target }}"
vars_files:
```

```
    vars/global_vars.yml
    roles:
    role: roles/functionality1
    when:

            enable_functionality1|default(true)

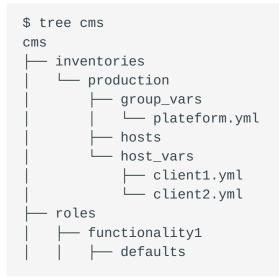
    role: roles/functionality2
    when:

                    enable_functionality2
                    default(false)
```

Don't forget to use the tags:

```
- name: "Config Management for {{ target }}"
hosts: "{{ target }}"
vars_files:
    vars/global_vars.yml
roles:
    role: roles/functionality1
    when:
        enable_functionality1|default(true)
    tags:
        functionality1
    role: roles/functionality2
    when:
        enable_functionality2|default(false)
    tags:
        functionality2
```

You should get something like this:



| | L main.yml | | L tasks | | main.yml | functionality2 | defaults | defaults | main.yml | tasks | main.yml | site.yml | yars | global_vars.yml

🖍 Note

You are free to develop your roles within a collection

8.4 Tests

Let's launch the playbook and run some tests:

```
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml
PLAY [Config Management for client1]
                           TASK [Gathering Facts]
* * * * * * * * *
       * * * * * * * * * * *
ok: [client1]
TASK [roles/functionality1 : Task in functionality 1]
                 * * * * * * * * * * * * * * * *
ok: [client1] => {
   "msg": "You are in functionality 1"
}
TASK [roles/functionality2 : Task in functionality 2]
         skipping: [client1]
PLAY RECAP
    * * * * * * * * * * * * *
                  * * * * * * * * * * * * * * * * * *
client1
                     : ok=<mark>2</mark>
                             changed=0 unreachable=0
                                                     failed=0
skipped=1 rescued=0 ignored=0
```

As you can see, by default, only the tasks of the functionality1 role are played.

Let's activate in the inventory the functionality2 for our targeted node and rerun the playbook:

```
$ vim inventories/production/host_vars/client1.yml
- - -
enable_functionality2: true
$ ansible-playbook -i inventories/production/hosts -e "target=client1" site.yml
PLAY [Config Management for client1]
                                 TASK [Gathering Facts]
      * * * * * * * * * * * * * * * *
                     * * * * * * * * * * *
ok: [client1]
TASK [roles/functionality1 : Task in functionality 1]
ok: [client1] => {
   "msg": "You are in functionality 1"
}
TASK [roles/functionality2 : Task in functionality 2]
  ok: [client1] => {
   "msg": "You are in functionality 2"
}
PLAY RECAP
                 client1
                   : ok=<mark>3</mark>
                           changed=0 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0
```

Try to apply only functionality2:

Let's run on the whole inventory:

```
$ ansible-playbook -i inventories/production/hosts -e "target=plateform"
site.yml
PLAY [Config Management for plateform]
                          TASK [Gathering Facts]
               * * * * * * * * * * *
ok: [client1]
ok: [client2]
TASK [roles/functionality1 : Task in functionality 1]
ok: [client1] => {
  "msg": "You are in functionality 1"
}
ok: [client2] => {
  "msg": "You are in functionality 1"
}
TASK [roles/functionality2 : Task in functionality 2]
ok: [client1] => {
  "msg": "You are in functionality 2"
}
skipping: [client2]
```

PLAY RECAP ************************************						
* * * * * * * * * * * * * * * * * * * *						
client1 skipped= <mark>0</mark>	rescued=0	: ok= <mark>3</mark> ignored= <mark>0</mark>	changed=0	unreachable=0	failed=0	
client2 skipped= <mark>1</mark>	rescued=0	: ok= <mark>2</mark> ignored=0	changed=0	unreachable=0	failed=0	

As you can see, functionality2 is only played on the client1.

8.5 Benefits

By following the advice given in the Ansible documentation, you will quickly obtain a:

- easily maintainable source code even if it contains a large number of roles
- a relatively fast, repeatable compliance system that you can apply partially or completely
- can be adapted on a case-by-case basis and by servers
- the specifics of your information system are separated from the code, easily auditable, and centralized in the inventory files of your configuration management.

9. Ansible - Working with filters

In this chapter you will learn how to transform data with jinja filters.

Objectives: In this chapter you will learn how to:

Transform data structures as dictionaries or lists;
 Transform variables.

🏁 ansible, jinja, filters

Knowledge: $\bigstar \bigstar \bigstar$ Complexity: $\bigstar \bigstar \bigstar \bigstar$

Reading time: 20 minutes

We have already had the opportunity, during the previous chapters, to use the jinja filters.

These filters, written in python, allow us to manipulate and transform our ansible variables.

Note

More information can be found here.

Throughout this chapter, we will use the following playbook to test the different filters presented:

```
- name: Manipulating the data
hosts: localhost
gather_facts: false
vars:
   zero: 0
   zero_string: "0"
   non_zero: 4
   true_booleen: True
   true_non_booleen: "True"
   false_boolean: False
```

```
false_non_boolean: "False"
  whatever: "It's false!"
  user_name: antoine
  my_dictionary:
    key1: value1
    key2: value2
  my_simple_list:
    - value_list_1
    - value_list_2
    - value_list_3
  my_simple_list_2:
    - value_list_3
    - value_list_4
    - value_list_5
  my_list:
    - element: element1
      value: value1
    - element: element2
      value: value2
tasks:
  - name: Print an integer
    debug:
      var: zero
```

Note

The following is a non-exhaustive list of filters that you are most likely to encounter or need. Fortunately, there are many others. You could even write your own!

The playbook will be played as follows:

```
ansible-playbook play-filter.yml
```

9.1 Converting data

Data can be converted from one type to another.

To know the type of a data (the type in python language), you have to use the type_debug filter.

Example:

```
- name: Display the type of a variable
  debug:
    var: true_boolean|type_debug
```

which gives us:

It is possible to transform an integer into a string:

```
    name: Transforming a variable type
debug:
var: zero|string
```

Transform a string into an integer:

```
    name: Transforming a variable type
debug:
var: zero_string|int
```

or a variable into a boolean:

```
name: Display an integer as a boolean debug:
var: non_zero | bool
name: Display a string as a boolean debug:
var: true_non_boolean | bool
name: Display a string as a boolean debug:
var: false_non_boolean | bool
```

```
- name: Display a string as a boolean
debug:
    var: whatever | bool
```

A character string can be transformed into upper or lower case:

```
    name: Lowercase a string of characters
debug:
var: whatever | lower
    name: Upercase a string of characters
debug:
var: whatever | upper
```

which gives us:

The replace filter allows you to replace characters by others.

Here we remove spaces or even replace a word:

```
name: Replace a character in a string
debug:
var: whatever | replace(" ", "")
name: Replace a word in a string
debug:
var: whatever | replace("false", "true")
```

which gives us:

The split filter splits a string into a list based on a character:

```
- name: Cutting a string of characters
debug:
   var: whatever | split(" ", "")
```

9.2 Join the elements of a list

It is frequent to have to join the different elements in a single string. We can then specify a character or a string to insert between each element.

```
name: Joining elements of a list
debug:
var: my_simple_list|join(",")
name: Joining elements of a list
debug:
var: my_simple_list|join(" | ")
```

which gives us:

9.3 Transforming dictionaries into lists (and vice versa)

The filters dict2items and itemstodict, a bit more complex to implement, are frequently used, especially in loops.

Note that it is possible to specify the name of the key and of the value to use in the transformation.

```
- name: Display a dictionary
  debug:
    var: my_dictionary
- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items
- name: Transforming a dictionary into a list
  debug:
    var: my_dictionary | dict2items(key_name='key', value_name='value')
- name: Transforming a list into a dictionary
  debug:
    var: my_list | items2dict(key_name='element', value_name='value')
TASK [Display a dictionary]
ok: [localhost] => {
    "my_dictionary": {
        "key1": "value1",
        "key2": "value2"
    }
```

```
}
TASK [Transforming a dictionary into a list]
* * * * * * * * * * * *
ok: [localhost] => {
    "my_dictionary | dict2items": [
        {
            "key": "key1",
            "value": "value1"
       },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}
TASK [Transforming a dictionary into a list]
                                                   * * * * * * * * * *
                      **********
ok: [localhost] => {
    "my_dictionary | dict2items (key_name = 'key', value_name = 'value')": [
        {
            "key": "key1",
            "value": "value1"
       },
        {
            "key": "key2",
            "value": "value2"
        }
    ]
}
TASK [Transforming a list into a dictionary]
* * * * * * * *
ok: [localhost] => {
    "my_list | items2dict(key_name='element', value_name='value')": {
        "element1": "value1",
        "element2": "value2"
    }
}
```

9.4 Working with lists

It is possible to merge or filter data from one or more lists:

```
- name: Merger of two lists
  debug:
    var: my_simple_list | union(my_simple_list_2)

ok: [localhost] => {
    "my_simple_list | union(my_simple_list_2)": [
        "value_list_1",
        "value_list_2",
        "value_list_3",
        "value_list_5"
    ]
}
```

To keep only the intersection of the 2 lists (the values present in the 2 lists):

Or on the contrary keep only the difference (the values that do not exist in the second list):

```
- name: Merger of two lists
  debug:
    var: my_simple_list | difference(my_simple_list_2)
```

}

If your list contains non-unique values, it is also possible to filter them with the unique filter.

```
- name: Unique value in a list
debug:
    var: my_simple_list | unique
```

9.5 Transformation json/yaml

You may have to import json data (from an API for example) or export data in yaml or json.

```
    name: Display a variable in yaml

 debug:
   var: my_list | to_nice_yaml(indent=4)
- name: Display a variable in json
 debug:
   var: my_list | to_nice_json(indent=4)
TASK [Display a variable in yaml]
ok: [localhost] => {
   "my_list | to_nice_yaml(indent=4)": "-
                                       element: element1\n value:
value1\n- element: element2\n value: value2\n"
}
TASK [Display a variable in json]
                                          ok: [localhost] => {
   "my_list | to_nice_json(indent=4)": "[\n {\n
                                                   \"element\":
\label{eq:lement1}, n \ \value': \value1\", n {\n }
\"element\": \"element2\",\n \"value\": \"value2\"\n }\n]"
}
```

9.6 Default values, optional variables, protect variables

You will quickly be confronted with errors in the execution of your playbooks if you do not provide default values for your variables, or if you do not protect them.

- 103/113 -

The value of a variable can be substituted for another one if it does not exist with the default filter:

}

Note the presence of the apostrophe ' which should be protected, for example, if you were using the shell module:

Finally, an optional variable in a module can be ignored if it does not exist with the keyword omit in the default filter, which will save you an error at runtime.

```
- name: Add a new user
ansible.builtin.user:
    name: "{{ user_name }}"
    comment: "{{ user_comment | default(omit) }}"
```

9.7 Associate a value according to another one (ternary)

Sometimes you need to use a condition to assign a value to a variable, in which case it is common to go through a set_fact step.

This can be avoided by using the ternary filter:

9.8 Some other filters

- {{ 10000 | random }} : as its name indicates, gives a random value.
- {{ my_simple_list | first }} : extracts the first element of the list.
- {{ my_simple_list | length }} : gives the length (of a list or a string).
- {{ ip_list | ansible.netcommon.ipv4 }} : only displays v4 IPs. Without dwelling on this, if you need, there are many filters dedicated to the network.
- {{ user_password | password_hash('sha512') }}: generates a hashed password in sha512.

10. Management server optimizations

In this chapter, we will review the configuration options that may be of interest to optimize our Ansible management server.

10.1 The ansible.cfg configuration file

Some interesting configuration options to comment on:

- Forks : by default to 5, it is the number of processes that Ansible will launch in parallel to communicate with remote hosts. The higher this number is, the more clients Ansible will be able to manage at the same time, and thus speed up processing. The value you can set is dependent on the CPU/RAM limits of your management server. Note that the default value, 5, is very small, the Ansible documentation states that many users set it to 50, even 500 or more.
- gathering: this variable changes the policy for collecting facts. By default, the value is implicit, which implies that facts will be collected systematically. Switching this variable to smart allows for collection facts only when they have not already been collected. Coupled with a facts cache (see below), this option can greatly increase performance.
- host_key_checking: Be careful with your server security! However, if you are in control of your environment, it can be interesting to disable the key control of remote servers and save some time at the connection. You can also, on remote servers, disable the use of the DNS of the SSH server (in /etc/ssh/sshd_config, option UseDNS no), this option wastes time at the connection and is, most of the time, only used in the connection logs.
- ansible_managed : This variable, containing Ansible managed by default, is typically used in file templates that are deployed on remote servers. It allows you to specify to an administrator that the file is managed automatically and that any changes they make to it will potentially be lost. It can be interesting to let the administrators have a more complete message. Be careful though, if you change this variable, it may cause daemons to restart (via the handlers associated with the templates).
- ssh_args = -C -o ControlMaster=auto -o ControlPersist=300s -o
 PreferredAuthentications=publickey: specify the ssh connection options. By disabling all authentication methods other than public key, you can save a lot of time. You can also increase the ControlPersist to improve performance (the documentation suggests that a value equivalent to 30 minutes may be appropriate). The connection to a client will stay open longer and can be reused when reconnecting to the same server, which is a significant time saving.
- control_path_dir : Specify the path to the connection sockets. If this path is too long, it can cause problems. Consider changing it to something short, such as / tmp/.cp.

• pipelining: Setting this value to True increases performance by reducing the number of SSH connections needed when running remote modules. You must first make sure that the requiretty option is disabled in the sudoers options (see documentation).

10.2 Caching the facts

Gathering facts is a process that can take some time. It can be interesting to disable this gathering for playbooks that don't need it (via gather_facts option) or to keep these facts in memory in a cache for a certain period of time (for example 24H).

These facts can be easily stored in a redis database:

```
sudo yum install redis
sudo systemctl start redis
sudo systemctl enable redis
sudo pip3 install redis
```

Don't forget to modify the ansible configuration:

fact_caching = redis
fact_caching_timeout = 86400
fact_caching_connection = localhost:6379:0

To check the correct operation, it is enough to request the redis server:

```
redis-cli
127.0.0.1:6379> keys *
127.0.0.1:6379> get ansible_facts_SERVERNAME
```

10.3 Using Vault

The various passwords and secrets cannot be stored in clear text with the Ansible source code, either locally on the management server or on a possible source code manager.

Ansible proposes using an encryption manager: ansible-vault.

The principle is to encrypt a variable or a whole file with the ansible-vault command.

Ansible will be able to decrypt this file at runtime by retrieving the encryption key from the file (for example) /etc/ansible/ansible.cfg. The latter can also be a python script or other.

Edit the /etc/ansible/ansible.cfg file:

```
#vault_password_file = /path/to/vault_password_file
vault_password_file = /etc/ansible/vault_pass
```

Store the password in this file /etc/ansible/vault_pass and assign necessary restrictive rights:

mysecretpassword

You can then encrypt your files with the command:

ansible-vault encrypt myfile.yml

A file encrypted by ansible-vault can be easily recognized by its header:

```
$ANSIBLE_VAULT;1.1;AES256
353765323436633533306131336638346261363162343239643337353633339613661326638396
6
6664322261633261356566383438393738386165333966660a34303266323334376263393631363
0
3437323012456166376630613465623538623332396433623933666165343366303663333436666
1
6434656630306261650a31336463626139393131373936393133666438653633376632626463333
0
6334
```

Once a file is encrypted, it can still be edited with the command:

ansible-vault edit myfile.yml

You can also deport your password storage to any password manager.

For example, to retrieve a password that would be stored in the rundeck vault:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import urllib.request
import io
import ssl
def get_password():
    1.1.1
    :return: Vault password
    :return_type: str
    1 + 1
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE
    url = 'https://rundeck.rockylinux.org/api/11/storage/keys/ansible/vault'
    req = urllib.request.Request(url, headers={
                           'Accept': '*/*',
                           'X-Rundeck-Auth-Token': '****token-rundeck****'
                          })
    response = urllib.request.urlopen(req, context=ctx)
    return response.read().decode('utf-8')
if __name__ == '__main__':
    print(get_password())
```

10.4 Working with Windows servers

It will be necessary to install on the management server several packages:

• Via the package manager:

sudo dnf install python38-devel krb5-devel krb5-libs krb5-workstation

and configure the /etc/krb5.conf file to specify the correct realms:

```
[realms]
ROCKYLINUX.ORG = {
    kdc = dc1.rockylinux.org
    kdc = dc2.rockylinux.org
}
```

[domain_realm]
.rockylinux.org = ROCKYLINUX.ORG

• Via the python package manager:

```
pip3 install pywinrm
pip3 install pywinrm[credssp]
pip3 install kerberos requests-kerberos
```

10.5 Working with IP modules

Network modules usually require the netaddr python module:

sudo pip3 install netaddr

10.6 Generating a CMDB

A tool, ansible-cmdb has been developed to generate a CMDB from ansible.

pip3 install ansible-cmdb

The facts must be exported by ansible with the following command:

```
ansible --become --become-user=root -o -m setup --tree /var/www/ansible/cmdb/
out/
```

You can then generate a global json file:

```
ansible-cmdb -t json /var/www/ansible/cmdb/out/linux > /var/www/ansible/cmdb/
cmdb-linux.json
```

If you prefer a web interface:

ansible-cmdb -t html_fancy_split /var/www/ansible/cmdb/out/

https://docs.rockylinux.org/