Ansible Playbooks

We have seen how to run single tasks or one time tasks using Modules, but what if you need to execute multiple tasks? Playbooks help to run them in a scripted way.

Playbooks define variables, configurations, deployment steps, assign roles, perform multiple tasks. For **E.g.** COPY / DELETE Files and Folders, install packages, start services. So primarily playbooks are defined to orchestrate the steps to multiple machines or servers and get them all to a certain desired state.

Playbook is written in YAML format with a .yml file extension. One needs to be very careful with the format and alignment which makes it very sensitive.

**It contains the following sections:**

1. Every playbook starts with 3 hyphens ‘—‘
2. **Host section** – Defines the target machines on which the playbook should run. This is based on the Ansible inventory file.
3. **Variable section** – This is optional and can declare all the variables needed in the playbook. We will look at some examples as well.
4. **Tasks section** – This section lists out all the tasks that should be executed on the target machine. It specifies the use of Modules. Every task has a name which is a small description of what the task will do and will be listed while the playbook is run.

**For Example,**

**If we need to install and configure Tomcat it will consist of the following tasks:**

1. Download and Install Tomcat
2. Configure Tomcat
3. Start Tomcat

**Similarly, another Example for usage of Tomcat used in the continuous delivery of DevOps, the tasks could be as follows:**

1. Stop application
2. Uninstall application
3. Install a new version of the WAR file.
4. Start application

**Sample Format of Playbook**

--- Playbook start

- hosts: webservers Specify the group or servers as per inventory to execute tasks

become: true

tasks:

- name: Copy Tomcat ZIP file to install location Short description of the task

copy: src=/home/ansible/niranjan/apache-tomcat-8.5.31.tar.gz dest=/opt/niranjan/tomcat

In the above script look at the alignment starting from the top and it has to be maintained else you will get syntax errors.

**To run any playbook use the following command**

**$ ansible-playbook <playbook.yml>**

**To check the playbook for syntax errors**

**$ ansible-playbook <playbook.yml> --syntax-check**

**To view hosts list**

**$ ansible-playbook <playbook.yml> --list-hosts**

Creating Playbooks With Examples

In this section, we will see multiple examples of how to create playbooks which you might need to run regularly. These playbooks will need to be created and run from the control machine.

Save all the below playbooks to a .yml file and run as shown below.

**$ ansible-playbook filename.yml**

**Example 1:** Create the file on the target machines or servers as mentioned in the inventory file and the webserver's group, save the below code with .yml extension and run the playbook.

- hosts: webservers

become: true

tasks:

- name: Create a file

file: path=/home/ansible/niranjan.txt state=touch

In the above example, we have used the **file**module to create the file.

**Example 2:** Create a directory with the mode as 775 and owner/group as Ansible.

---

- hosts: webservers

become: true

tasks:

- name: Create directory

file: path=/home/ansible/niranjan state=directory mode=775 owner=ansible group=ansible

**Example 3:**Create multiple directories. To create multiple directories with one single task you can use the loop **with\_items** statement. So when you run the below playbook it is interpreted as 3 different tasks.

---

- hosts: webservers

become: true

tasks:

- name: Create multiple directories

file: path={{item}} state=directory

with\_items:

- '/home/ansible/vn1'

- '/home/ansible/vn2'

- '/home/ansible/vn3'

**Example 4:**Create a user. Let’s look at the **user**module to create and delete users in the playbook.

---

- hosts: webservers

become: true

tasks:

- name: Create User

user: name=niranjan password=niranjan groups=ansible shell=/bin/bash

**Example 5:** Remove user. Removing a user is very easy and it will need the **state** to be set to **absent**. This is equivalent to the **userdel** command in Linux.

---

- hosts: webservers

become: true

tasks:

- name: Remove User

user:

name=niranjan state=absent remove=yes force=yes

In the above playbook, **remove=yes** will remove the home directory and **force=yes** will remove the files in the directory.

**Example 6:** Copy content to a file using the copy module.

If you need to copy a file to the target machines or servers use the **src** and **dest** in the copy module.

---

- hosts: webservers

become: true

tasks:

- name: Copy content to file

copy: content="Hello World Niranjan \n" dest=/home/ansible/niranjan.txt

**For Example,**

copy: src=/home/ansible/niranjan.txt dest=/tmp/niranjan.txt

**Example 7:**Replace all instances of a string.

Using **replace**module we can replace a word with another word. The replace module will need 3 parameters i.e. ‘path’, ‘regexp’ (to find the particular word) and ‘replace’ (providing another word for replacement).

- hosts: webservers

tasks:

- name: Replace example

replace:

path: /home/ansible/niranjan.txt

regexp: 'hello'

replace: "world"

**Example 8:**Archive or ZIP files and Folders

Using the Ansible **archive**module you can compress files or folders to ‘zip’, ‘.gz’, or ‘bz2’ format.

**Note**: **The files or folders to be compressed should be available on the target servers and should have the packages for tar, bzip2, gzip, zip file installed on them. You can have a separate playbook task for installing these packages.**

---

- hosts: all

become: true

tasks:

- name: Ansible zip file example

archive:

path: /home/ansible/niranjan.txt

dest: /home/ansible/niranjan.zip

format: zip

The above playbook will zip the file niranjan.txt to niranjan.zip file

---

- hosts: all

tasks:

- name: Ansible zip multiple files example

archive:

path:

- /home/ansible/niranjan1.txt

- /home/ansible/niranjan2.txt

dest: /home/ansible/niranjan.zip

format: zip

The above playbook will zip multiple files to niranjan.zip file.

- hosts: all

tasks:

- name: Ansible zip directory example

archive:

path:

- /home/ansible

dest: /home/ansible/niranjan.zip

format: zip

The above playbook will zip all files in the /home/ansible directory.

**Example 9:**Working with date and timestamp

Using the system date and timestamp helps in certain status or logging purposes. The Ansible facts provide access to remote or target servers date and time. So we can use the **debug module**to print the output along with the **var**attribute as shown below.

---

- hosts: webservers

become: true

tasks:

- name: Date and Time Example in Ansible

debug:

var=ansible\_date\_time.date

The above playbook displays the date.

---

- hosts: webservers

become: true

tasks:

- name: Date and Time Example in Ansible

debug:

var=ansible\_date\_time.time

The above playbook displays the time.

- hosts: all

tasks:

- name: Ansible timestamp filename example

command: touch niranjan{{ansible\_date\_time.date}}.log

The above playbook will create a dynamic file based on the current date for **E.g**. niranjan2018-07-15.log

**Example 10:**Variables Example

Variables are used to store values. In the below Example I am declaring the variable **name** with value **niranjan**. The output will be **niranjan**.

- hosts: all

vars:

name: niranjan

tasks:

- name: Ansible Basic Variable Example

debug:

msg: "{{ name }}"

We can also have an array or a list of variables as in the below **Example**.

- hosts: all

vars:

name:

- Vasudevamurthy

- Niranjan

tasks:

- name: Ansible Array Example

debug:

msg: "{{ name[1] }}"

The indexing of the array starts from ZERO (0). Hence the output in the above example will be Niranjan.

**Example 11:** Register Variables

We can also capture the output of any task to a register variable.

- hosts: all

tasks:

- name: Ansible register variable basic example

shell: "find \*.txt"

args:

chdir: "/home/Ansible"

register: reg\_output

- debug:

var: reg\_output

**Note: To display – use the msg attribute and to capture any value use the var attribute in the – debug module**

**Example 12:** Playbook to install vim editor and GIT on the target servers or machines.

In this playbook, we have made use of the **yum**module to install the latest version of the software packages.

---

- hosts: webservers

become: true

tasks:

- name: Install Package

yum: name=vim,git state=latest

**Example 13:** Install Apache server. Save the below code and run playbook as shown below.

---

- hosts: webservers

become: true

tasks:

- name: Install Package

yum: name=httpd state=present

- name: Start httpd service

service: name=httpd state=started

Apart from the **yum**module, the **service**module is also used to start the httpd service. The tasks run from top to bottom synchronously.

**Example 14:**Install JDK

The following playbook will automate to install JDK 8 on all target machines or servers. JDK is a pre-requisite for most of the other software packages like Maven or Tomcat.

---

- hosts: webservers

become: true

vars:

download\_url: http://download.oracle.com/otn-pub/java/jdk/8u171-b11/512cd62ec5174c3487ac17c61aaa89e8/jdk-8u171-linux-x64.rpm

tasks:

- name: Download JDK 8 RPM file

command: "wget --no-check-certificate --no-cookies --header 'Cookie: oraclelicense=accept-securebackup-cookie' {{download\_url}} "

- name: Install JDK 8

command: "rpm -ivh jdk-8u171-linux-x64.rpm"

**Example 15:**Install Maven

The tasks performed are to download the maven file from the URL using the **get\_url** module, extract the file downloaded, move it to a smaller directory, update and run the profile where the maven is added to the path.

---

- hosts: webservers

become: true

tasks:

- name: Download Maven

get\_url: url=http://www-us.apache.org/dist/maven/maven-3/3.5.3/binaries/apache-maven-3.5.3-bin.tar.gz dest=/opt/niranjan/apache-maven-3.5.3-bin.tar.gz

- name: Extract Maven

command: tar xvf /opt/niranjan/apache-maven-3.5.3-bin.tar.gz -C /opt/niranjan

- name: Move to a smaller directory

command: mv /opt/niranjan/apache-maven-3.5.3 /opt/niranjan/maven

- name: Update Profile

copy: content="export M2\_HOME=/opt/niranjan/maven \n" dest=/etc/profile.d/maven.sh

# lineinfile is used to add additional or append lines to existing files.

- lineinfile:

path: /etc/profile.d/maven.sh

line: 'export PATH=${M2\_HOME}/bin:${PATH}'

- name: Source profile

shell: source /etc/profile.d/maven.sh

**Example 16:**Install Tomcat 8

The below playbook helps to install and start Tomcat 8 on to the target machines or servers.

You can click [here](http://tomcat.apache.org/download-80.cgi" \t "_blank) to copy the link location of the latest version of Tomcat 8. Click [here](http://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.32/bin/apache-tomcat-8.5.32.tar.gz" \t "_blank) for the URL containing Tomcat 8 tar file that I have used in this playbook.

---

- hosts: webservers

become: true

gather\_facts: no

tasks:

- name: Download Tomcat

get\_url: url=http://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.32/bin/apache-tomcat-8.5.32.tar.gz dest=/home/ansible

- name: Extract the file downloaded tomcat file

command: tar xvf apache-tomcat-8.5.32.tar.gz

- name: Move the Tomcat directory to a smaller one

command: mv apache-tomcat-8.5.32 tomcat

- name: Change Ownership and group of the Tomcat directory

file: path=/home/ansible/tomcat owner=ansible group=ansible mode=775 state=directory recurse=yes

- name: Start Tomcat

command: nohup /home/ansible/tomcat/bin/startup.sh # Execute command even after you have exited from the shell prompt

become: true

become\_user: ansible

**Example 17:** pre\_tasks, post\_tasks, and tags

You can use **pre\_tasks and post\_tasks**to run certain tasks before or after running the main task.

Normally in a playbook, you have so many tasks that are executed. What if you need to execute only a certain task? Tags are the answer to it. Let’s look at the below option which has all the 3 options. It has 2 tasks i.e. one with a TAG and one without a TAG.

---

- name: Pre , Post tasks and Tags example

hosts: localhost

become: true

tags:

- niranjan

pre\_tasks:

- debug: msg="Started task with tag - niranjan.

tasks:

- name: Going to execute the main task

debug: msg="Currently in the target server"

post\_tasks:

- debug: msg="Completed task with tag - niranjan.

- name: Play without tags

hosts: localhost

become: true

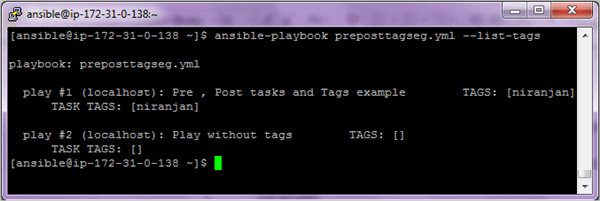
tasks:

- name: Command to list files

shell: ls -lrt > niranjan.txt

Let’s see what happens while running the playbook with the –list-tags option

**$ ansible-playbook preposttagseg.yml --list-tags**

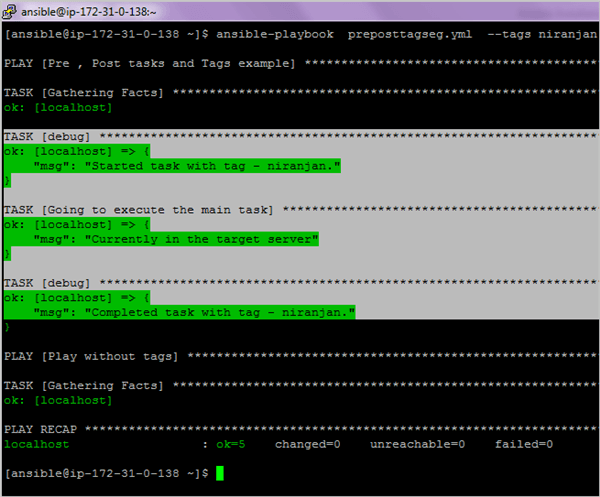


The output above looks better and clear. Play#1 has a tag niranjan but Play#2 does not have any tags.

If you need to execute the tasks with the tag niranjan then the command to run would be:

**$ ansible-playbook preposttagseg.yml --tags niranjan**

The second play is not executed and the file is not created.



**Example 18:**Handlers

Any software package will have configuration files and any changes to it will have effect only when the service is restarted. So you need to have the service set to restart. For **E.g.** In the below playbook if you run it multiple times the service will restart anyway irrespective of the changes done or not, which is not correct.

---

- hosts: webservers

tasks:

- name: Install the apache Package

yum: name=httpd state=latest

- name: Copy httpd configuration file

copy: src=/home/ansible/httpd.final dest=/etc/httpd/conf/httpd.conf

- name: Copy index.html file

copy: src=/home/ansible/index.html dest=/var/www/html

**# This service below is executed irrespective of changes done or not to any config files**

- name: Start and Enable httpd service

service: name=httpd state=**restarted** enabled=yes

So we need to restart service only if the changes are done to configuration files. **Handlers** provide that feature.

So the proper flow with handlers would be to have a **notify** option.

---

- hosts: webservers

become: true

tasks:

- name: Install httpd package

yum: name=httpd state=latest

- name: Copy the httpd configuration file

copy: src=/home/ansible/httpd.final dest=/etc/httpd/conf/httpd.conf

- name: Copy index.html file

copy: src=/home/ansible/index.html dest=/var/www/html

notify:

- restart httpd

- name: Start httpd service

service: name=httpd state=started enabled=yes

handlers:

- name: restart httpd

service: name=httpd state=restarted

So for the first time, Apache server will be installed and started. Even if you re-run the playbook without any changes done the httpd service will not restart as it is already started.

If there are any changes to the config files or if the HTML files are changed then once the playbook is run the handler is notified to restart the service. The name in the notify section and handlers should be the same. The handler is written like any other task but is called only if there are changes.

Ansible Vault

Most of the times when sensitive or confidential data need to be protected in the playbook, then it can be encrypted rather than just keeping it in a text file which is readable by all. Ansible Vault allows you to encrypt the playbook to protect the confidential data.

**For Example, consider the following task where a confidential job agreement is being copied.**

In such cases, you would need an Ansible Vault.

---

- hosts: webservers

become: true

tasks:

- name: Copying Confidential Job Agreement

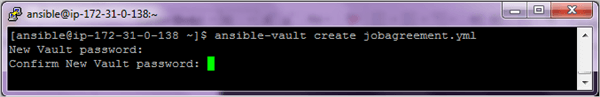
copy: content="This is a Confidential Job Agreement" dest=/home/ansible/jobagreement.txt

**Following are the steps that you need follow to encrypt the above playbook files.**

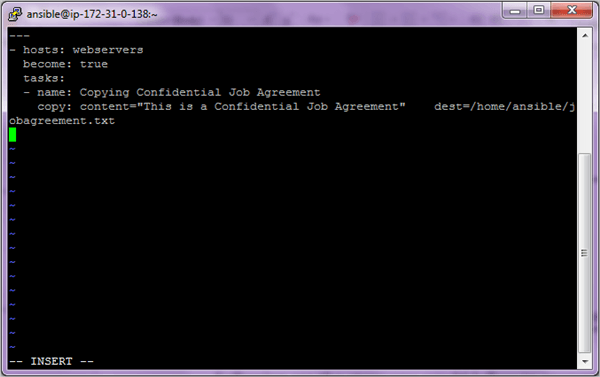
**#1) Creating new encrypted files**

To create new encrypted files with vault use the **ansible-vault create**command.

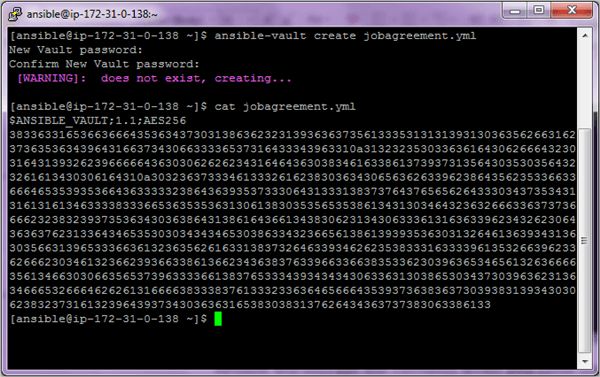
**$ ansible-vault create jobagreement.yml**



After confirming password an editing window will open to add contents to the file.



Ansible will encrypt the contents when you close the file. Instead of seeing the actual contents you will see encrypted blocks.



**#2) To encrypt an existing yml file use the following**

**$ ansible-vault encrypt existingfile.yml**

Password will again be asked for encryption.

**#3) Viewing encrypted file**

Use the command **ansible-vault view**to look at the actual contents of the file.

**$ ansible-vault view jobagreement.yml**

You will be asked for the password again to look at the contents of the file.

**#4) Editing encrypted files**

If you need to edit the file use the command **ansible-vault edit**

**$ ansible-vault edit users.yml**

Enter the password to edit the file.

**#5) Changing password of the encrypted files**

Use the command **ansible-vault rekey**to change the password of the file.

**$ ansible-vault rekey jobagreement.yml**

**#6) Run an encrypted Ansible playbook file**

Use the option –ask-vault-pass with the ansible-playbook command.

**$ ansible-playbook users.yml --ask-vault-pass**

**#7) Manually decrypting the encrypted files**

Use the command ansible-vault decrypt command.

**$ ansible-vault decrypt jobagreement.yml**

Summary

Well in this tutorial, we saw the two most important aspects of configuration management which are **Ansible Playbooks and protecting sensitive data using Ansible Vaults.**