**Configuration Management Tool (Ansible)**

* open-source IT automation engine
* improve the scalability, consistency, and reliability of your IT environment.

**what can it automate?**

* Provisioning: Set up the various servers you need in your infrastructure.
* [Configuration management](https://cloudacademy.com/learning-paths/configuration-management-tools-31/): Change the configuration of an application, OS, or device; start and stop services; install or update applications; implement a security policy; or perform a wide variety of other configuration tasks.
* [Application deployment](https://cloudacademy.com/lab/ansible-deploying-web-applications/): Make [DevOps](https://cloudacademy.com/what-is-devops/) easier by automating the deployment of internally developed applications to your production systems.

Ansible is an open source configuration management and orchestration utility. It can automate the configuration of remote hosts are virtual machines using ansible it is possible to lunches or shutdown multi-tiered architecture due to this reason ansible is used for performing rolling updates with 0% downtime. Instead of creating shell scripts for management remote servers ansible uses **plays.** A play is collection of task that should be performed on remote host a file which contains these plays are called as ansible playbooks.

Ansible uses agent less architecture i.e. ansible pushes its tasks via ssh so it does not require any agent to be installed on the manage hosts.

**Note:**

Ansible requires python to be installed on all the machines

**Controller:** This is the centre on which we install ansible

**Managed Hosts or managed Nodes:** these are the remote machines which will be controlled by ansible.

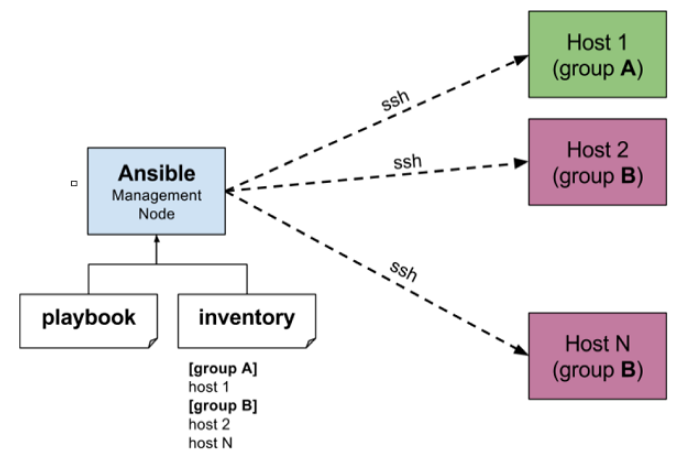
**Disadvantages of Ansible:**

It cannot perform installation of the basic o/s.

It cannot monitor the changes done on the remote host.

**How Ansible works?**

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



**Configure and Installation of Ansible**

1 Create 3-5 linux vm's or aws instances

2 The main machine where ansible is installed is called controller the remaining remote servers that ansible configures are called manged hosts.

3 Establish passwordless ssh between the controller and the managed hosts

4 To install ansible

a) Open terminal in

b) Update the apt repository

sudo apt-get update

c) Install software-properties-common

sudo apt-get install -y software-properties-common

d) Add the latest version of ansible ppa to apt repository

sudo apt-add-repository ppa:ansible/ansible

e) Update the apt repository

sudo apt-get update

f) Install ansible

sudo apt-get install -y ansible

5 To check the version of ansible

ansible –version

Note: Install the same version of python in all managed and controller nodes

Ansible store all the remote managed hosts info in a file called as inventory file.

The default location of thsi file is

/etc/ansible/hosts

Open this file and add the ipaddress of all the managed hosts

sudo vim /etc/ansible/hosts

10.10.10.72

10.10.10.73

10.10.10.74

10.10.10.75

Save and quit

**Modules in ansible**

Ansible performs remote configuration of servers by using in built modules. These have been created using python. Each module is deigns for performing a specific task

**Important modules in ansible**

**command** : This is the default module of ansible and it is used to fire linux commands on the remote managed nodes

**shell** : This is used for running shell scripts and linux commands that involve redirection and piping

**ping** : This is used to check if a remote server is pingable or not

**copy** : Used for copying files from the controller to the managed nodes

**fetch** : Used to copying files from the managed nodes to the controller

**file** : Used for creating files or directories on the managed nodes

**user** : Used for user management on the managed hosts i.e creating, modifying and deleting users.

**apt** : Used for package management on the managed nodes i.e installing, upgrading, uninstalling etc. It works on ubuntu, debain based linux flavors.

**yum** : similar to apt but it works on redhat linux, centos, fedoraOEL etc linux flavours.

**git** : This is used for performing git version controlling on the managed nodes

**debug** : This is used for displaying the output of a module

**service** : this is used to start,restart or stop services

**uri** : This is used to check if a rmeote url is reachable or not .

**stat** : this is used to check the info about files and directories on the managed nodes.

**ec2 : This is used for creating ec2 instances on the aws cloud**

**docker\_image** : This is used for executing the commands of docker that are related to docker images

**docker\_container** : This is used for container management on the managed nodes

**docker\_login** : Used for loginin into hub.docker.com from the managed nodes.

**setup:** This is used to capture system facts that is system information.

**Command module**

Ansible command to see the memory info on all manged nodes

ansible all -m command -a 'free'

Ansible command to create an empty file on all the managed nodes

ansible all -m command -a 'touch file1'

**Shell Module**

ansible command to execute ls -la on managed nodes and save the output into a file called file1 on the managed node

ansible all -m shell -a 'ls -la > file1'

Ansible command to install docker on all managed nodes

ansible all -m shell -a 'curl -fsSL https://get.docker.com -o get-docker.sh'

ansible all -m shell -a 'sh get-docker.sh'

**Copy Module**

ansible command to copy a file into all the managed node

ansible all -m copy -a 'src=myinventory dest=/tmp'

ansible command to copy a file and also change its ownership groupownership and premissions

ansible all -m copy -a 'src=myinventory dest=/tmp owner=Anu group=root mode=700' -b

Note: -b stands for become. It is used for giving higher previlages to execute a command on the managed nodes

**File Module**

This is used for creating files and directories on the managed nodes

ansible command to create a file in /tmp on all nodes

ansible all -m file -a 'name=/tmp/file state=touch'

**Note**: state=touch is used for creating files

state=directory is used for creating directories

state=absent is used for deleting files/directories

Ansible command to create a directory in /tmp folder

ansible all -m file -a 'name=/tmp/dir1 state=directory'

File module can also be used for changing the permissions of files and also the ownership and groupownership of files

ansible all -m file -a 'name=/tmp/file1 state=touch owner=Anu group=root mode=700' -b

**apt module**

This is used for package management on ubuntu based machines

To install tree on all the managed nodes

ansible all -m apt -a 'name=tree state=present' -b

**Note**: state=present is for install

state=absent is for uninstall

state=latest is for upgrading to latest version

To uninstall git from all managed nodes

ansible all -m apt -a 'name=git state=absent' -b

To install apache2 after updating the apt repository on all managed nodes

ansible all -m apt -a 'name=apache2 state=present update\_cache=yes' -b

**Service module**

This is used for starting stopping and restarting services

Ansible command to stop tomcat7 on all managed nodes

ansible all -m service -a 'name=tomcat7 state=stopped' -b

*state= strarted for starting the service*

*state=stopped for stopping the service*

*state=restarted for restarting the service*

**uri Module**

This is used to check if a remote url is reachable or not

ansible command to check if google.com is reachable from all managed nodes

ansible all -m uri -a 'url=http://google.com status=200'

status=200 is pass status

status=-1 is failure status

**Git module**

This is used for git version controlling on the managed nodes

Install git on all managed nodes and download a remote git repository

ansible all -m apt -a 'repo=https://github.com/selenium-saikrishna/maven.git dest=/tmp/mygit'

**Setup module**

This is used to capture system facts that is system information.

$ ansible all –m setup

To find information about one variable we can give that variable name as on argument

Ext; to find the kernel on remote machine is running

$ ansible 192.168.60.101 –m steup –a ‘filter=facter\_kernel’

**Fetch module**

To fetch a file from manage nodes into the controller machine we can use fetch module.

$ ansible all –m fetch –a ‘src=/home/vagrant/file1 dest=/home/vagrant’

**Configure the managed nodes**

Configuring the managed nodes in three defferent ways

1 Adhoc commands

2 Playbooks

3 Roles

**Adhoc Commands**

**Syntax of adhoc commands**

ansible all/groupname/ipaddress -i /etc/ansible/hosts -m command -a 'arguments'

-i is not manadatory when we use the default inventory file ie /etc/ansible/hosts

-m is not manadatory when we use the deafult module ie command module

**Configuing apache2**

Install apache2 on one mnaaged node and edit the content of its default index.html file,Restart apache2 and check if this is reachable from all other managed nodes

Install apache2 on one managed node

ansible 10.10.10.72 -m apt -a 'name=apache2 state=present cache\_busting=yes' -b

Edit the index.html file

ansible 10.10.10.72 -m copy -a 'content="Hello Aianalytics\n" dest=/var/www/html/index.html' -b

Restart the apache2 service

ansible 10.10.10.72 -m service -a 'name=apache2 state=restarted' -b

Check the url response

ansible all -m uri -a 'url=http://10.10.10.72 status=200'

**Ansible playbook**

The ansible commands that we have executed till now are called has ad-hoc commands and they are useful for only performing single operation and they can work only on a single setup arguments. they cannot be used for complex configuration activities.

Ad-hoc commands can execute only one module at time to work on multiple modules we can use playbooks.

Playbooks are created in yamil format

Playbooks are powerful and flexible for performing cm

Using playbooks we can change lengthy and complex at administrative activities in to repeatable routines

Ansible playbooks are combination of plays each play defines set of operation that should be performed on manage nodes these operation are called as tasks and managed nodes called as hosts.

Each task execute specific modules the modules are executed in the order In which we present in the playbooks.

**Ansible playbook for isntalling tree on all managed nodes**

vim playbook1.yml

---

- name: Installing tree

hosts: all

tasks:

- name: tree installation

apt:

name: tree

state: present

- name: git install

apt:

name: git

state: present

Save and quit

To execute the playbook

ansible-playbook playbook1.yml -b

**Ansible playbook for user creation**

Vim plavybook2.yml

---

- name: User Creation

hosts: all

tasks:

- name: Creating users

user:

name: Lakshmi

password: aianalytics

home: /home/vagrant/Lakshmi

shell: /bin/bash

...

Save and quit

To execute the playbook

ansible-playbook playbook2.yml -b

**Ansible playbbok for craeting directory and copy passwd file to that dirctory on all managed nodes**

- name: Creating dir and copying the passwd file

hosts: all

tasks:

- name: Creating dir

file:

name: /tmp/newdir

state: directory

- name: Copying passwd file

copy:

src: /etc/passwd

dest: /tmp/newdir/passwd

...

**Ansible playbbok for installing git and download from github repository**

---

- name: Install git and download a remote repo

hosts: all

tasks:

- name: Install git

apt:

name: git

state: present

- name: Download the remote repo

git:

repo: https://github.com/selenium-saikrishna/maven.git

dest: /tmp/git1

...

**Create an ansible playbook for going to the manage nodes and fetching the all the users who are using /bin/bash shell**

---

- name: for capturing user eiht /bin/bash/ shell

hosts: all

tasks:

- name: Capturing info from /etc/password

shell: grep /bin/bash /etc/passwd > file1

- name: Fethcing files

fetch:

src: /home/vagrant/file1

dest: /home/vagrant

....

**Install apche2 on one manage node start apache2 service and check the url response.**

---

- name: creating users and capturing username and home dir

hosts: 192.168.60.2

become: yes

tasks:

- name: update apt repo and install apache2

apt:

name: apache2

state: present

update\_cache: yes

- name: starting apache service

service:

name: apache2

state: started

- name: checking url response

uri:

url: http://192.168.60.101

status: 200

...

**Create a ansible playbook installing apache2 one more mcahine**

---

- name: installing apache2

hosts: 192.168.60.1

tasks:

- name: installing apache2

apt:

name: apache2

state: present

- name: installing git

hosts: 192.168.60.2

tasks:

- name: installing git

apt:

name: git

state: present

...

**Note**: the playbooks we have created till now are working only on group hosts that is the entire paly is getting executed on that group of hosts. To perform different activity’s on different group hosts we can create multiple plays in playbook.

**Variables**

Variables are classified into 3 types

1) **Global scope**: these variables are defined from the command prompt and they can affect the complete playbook.

2) **Host scope**: these variables are defied with respect the hosts and it can affect all the plays with in the

3) **Play scope**: These are defined at the level of induvial play and they can effect only that particular play.

**Global Scope Variables**

Passing the global scope variables through playbook command execution by using –extra-vars keyword

Create an ansible playbook which can be used either for installing or uninstalling packages similarly it should also be used either for updating or non-updating the apt repository by using global scope variables

---

- name: installing/uninstalling packages

hosts: all

tasks:

- name: installing/uninstalling packages

apt:

name: "{{a}}"

state: "{{b}}"

update\_chache="{{c}}"

...

To run this playbook command prompt for installing maven without updating the repository:

$ ansible-playbook playbook12.yml --extra-vars “a=maven b=present c=no” -b

Similarly we can use the same the playbook for uninstalling git after updating apt repository.

$ansible-playbook playbook12.yml – extra-vars “a=git b=absent c=yes” –b

**Create an ansible playbook for creating files or directories on the manage nodes and also for controlling the ownership groupowenership and permissions by passing values through global scope variable**

Vim playbook13.yml

---

- name: creating files and directories

hosts: all

tasks: file creating/ dir creation and controlling ownership

file:

name: "{{a}}"

state: "{{b}}"

owner: "{{c}}"

group: "{{d}}"

mode: "{{e}}"

...

Vim playbook14.yml

- name: creating files and directories

hosts: all

tasks: file creating/ dir creation and controlling ownership

file:

name: "{{name}}"

state: "{{state}}"

owner: "{{owner}}"

group: "{{group}}"

mode: "{{mode}}"

...

$ ansible-playbook playbook13.yml - - extra-vars “a=file1 b=touch c=root d=sai e=111” -b

$ ansible-playbook playbook14.yml - - extra-vars “name=dir1 state=directory owner=vagrant group=sai mode=777” -b

using this play book we can create directory and controller the ownership and group ownership

**Note**: global scope have the highest presidency compare host scope and play scope.

**Play scope variables**

These variables can effect only that particular play and they should be defined with in the playbook

Create an ansible playbook for installing apache2 on all manage nodes after updating the apt repository

---

- name: package installation/uninstallation

hosts: all

vars:

- a: apache2

- b: present

- c: yes

tasks:

- name: installing/uninstallation

apt:

name: "{{a}}"

state: "{{b}}"

update\_chache: "{{c}}"

...

Note: if we execute above playbook the variable values arr taken from variable section in this playbbok but we overide these variable with global scope variables since blobal scope variable have high priority

**Host Scope Variables**

These are categorised into 2 types

1 Variables to work on a group of hosts

2 Variables to work on a single host

**Variable to work on a group of hosts**

1 Change dir to the fodler where all playbooksare present

cd path\_of\_folder\_where\_playbooks\_are\_present

2 Create a new dir "group\_vars"

mkdir group\_vars

3 Create a file whose name should be similar tothe group name in our inventory file

vim webserver

---

a: Radha

b: aianalytics

c: /home/vagrant/Radha

...

4 Change dir back to the fodler where the playbooks are present

cd ..

5 Create a playbook for user creation on the webserver group

vim playbook10.yml

---

- name: User Creation

hosts: webserver

tasks:

- name: Creating users

user:

name: "{{a}}"

password: "{{b}}"

home: "{{c}}"

...

ansible-playbook playbook10.yml -b

**Variables to work on a single host**

1 Change dir to the folder where all playbooks are present

cd path\_of\_playbooks\_folder

2 Create a folder "host\_vars"

mkdir host\_vars

3 Change dir to host\_vars folder

cd host\_vars

4 Create a file whose name is same as an ipaddress of one machine in the inventory file

vim 10.10.10.74

---

a: tree

b: present

c: no

...

5 vim playbook11.yml

- name: Installing tree

hosts: 10.10.10.74

tasks:

- name: Install tree

apt:

name: "{{a}}"

state: "{{b}}"

update\_cache: "{{c}}"

...

**Variable Priorities**

1 Global scope

2 Host Scope - single machine (host\_vars)

2 Host Scope - Group of machines (group\_vars)

4 Play scope varibles

**Loops in ansible**

Loops can be implemented in ansible in 2 ways

1 with\_items

2 with\_sequence

with\_items is used to pass data to the module and depending on how many sets of data are present with\_items will loop those many number of times with\_sequence is used to run a module specific number of times based on a count

**ansible playbook to install multiple s/w packages**

vim playbook12.yml

---

- name: Install s/w

hosts: 10.10.10.74

tasks:

- name: Install s/w packages

apt:

name: "{{item}}"

state: present

update\_cache:

with\_items:

- tree

- git

- apache2

...

**Playbook for creating multiple users and copying some files into the users home dir**

- name: Creating user and copying files

hosts: all

tasks:

- name: User Creation

user:

name: "{{item.a}}"

password: "{{item.b}}"

home: "{{item.c}}"

with\_items:

- {a: Babu,b: aianalytics,c: /home/Babu}

- {a: Sravani,b: aianalytics,c: /home/Sravani}

- name: Copying files

copy:

src: "{{item.a}}"

dest: "{{item.b}}"

owner: "{{item.c}}"

group: "{{item.d}}"

with\_items:

- {a: /etc/passwd,b: /home/Babu,c: Babu,d: Sravani}

- {a: /etc/group,b: /home/Sravani,c: Sravani,d: Babu}

- Name: Handlers example playbook

**Handlers:**

Handlers are just like regular tasks in an Ansible playbook (see Tasks) but are only run if the Task contains a notify directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Example:

---

Hosts: all

Tasks:

-name: install apache2

Apt;

Name: apache2

State: present

Update\_cache: yes

Notify: restart\_apache2

Handlers:

-name: restart apache2

Service:

Name: apache2

State: restarted

…

**Note:** Handlers are executed if any changes are done in calling tasks. We can mention Handlers in any order but they are executed in order specified in handler section

**Error Handling**

Blocks also introduce the ability to handle errors in a way similar to exceptions in most programming languages. Blocks only deal with ‘failed’ status of a task. A bad task definition, an undefined variable or an unreachable host are not *rescuable* errors.

Block error handling example¶

tasks:

**-** name**:** Handle the error

block**:**

- debug:

msg**:** 'I execute normally'

**-** name**:** i force a failure

command**:** /bin/false

- debug:

msg**:** 'I never execute, due to the above task failing, :-('

rescue**:**

- debug:

msg**:** 'I caught an error, can do stuff here to fix it, :-)'

This will ‘revert’ the failed status of the task for the run and the play will continue as if it had succeeded.

There is also an always section, that will run no matter what the task status is.

Block with always section

**-** name**:** Always do X

block**:**

- debug:

msg**:** 'I execute normally'

**-** name**:** i force a failure

command**:** /bin/false

- debug:

msg**:** 'I never execute :-('

always**:**

- debug:

msg**:** "This always executes, :-)"

They can be added all together to do complex error handling.

Block with all sections¶

**-** name**:** Attempt and graceful roll back demo

block**:**

- debug:

msg**:** 'I execute normally'

**-** name**:** i force a failure

command**:** /bin/false

- debug:

msg**:** 'I never execute, due to the above task failing, :-('

rescue**:**

- debug:

msg**:** 'I caught an error'

**-** name**:** i force a failure in middle of recovery! >:-)

command**:** /bin/false

- debug:

msg**:** 'I also never execute :-('

always**:**

- debug:

msg**:** "This always executes"

The tasks in the block would execute normally, if there is any error the rescue section would get executed with whatever you need to do to recover from the previous error. The always section runs no matter what previous error did or did not occur in the block and rescue sections. It should be noted that the play continues if a rescue section completes successfully as it ‘erases’ the error status (but not the reporting), this means it won’t trigger max\_fail\_percentage nor any\_errors\_fatal configurations but will appear in the playbook statistics.

Another example is how to run handlers after an error occurred :

Block run handlers in error handling¶

tasks:

**-** name**:** Attempt and graceful roll back demo

block:

- debug:

msg**:** 'I execute normally'

notify**:** run me even after an error

**-** command**:** /bin/false

rescue:

**-** name**:** make sure all handlers run

meta**:** flush\_handlers

handlers:

**-** name**:** run me even after an error

debug:

msg**:** 'This handler runs even on error'

**Ansible Roles**

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Each role is basically limited to a particular functionality or desired output, with all the necessary steps to provide that result either within that role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

**Create and Configuring Roles**

cd /etc/ansible/roles

ansible-galaxy init apache --offline

**Note:** role is created with apache after executing above command. Using tree see the role stracture(tree apache)

1 Creating tasks for apache2

a)cd apache/tasks

b) sudo vim main.yml

---

- include: install.yml

- include: configure.yml

- include: check\_url\_response.yml

...

Save and quit (:wq)

c) sudo vim install.yml

---

- name: Install apache2

apt:

name: apache2

state: present

update\_cache: yes

...

Save and quit(:wq)

d) sudo vim configure.yml

---

- name: Send index.html file

copy:

src: index.html

dest: /var/www/html/index.html

notify:

restart\_apache2

...

Save and quit(:wq)

e) sudo vim check\_url\_response.yml

---

- name: check\_url\_reponse

uri:

url: http://10.10.10.74

status: 200

...

Save and quit (:wq)

2 Creating the static file that should be copied to remote

managed nodes

cd ..

cd files

sudo vim index.html

<html>

<body>

<h1>Aianalytics</h1>

</body>

</html>

Save and quit(:wq)

3 Create the handlers

cd ..

cd handlers

sudo vim main.yml

---

- name: restart\_apache2

service:

name: apache2

state: restarted

...

Save and quit(:wq)

4 Create a playbook to call this role

cd ..

cd ..

sudo vim configue\_apache2.yml

---

- name: Configuring apache2

hosts: all

roles:

- apache

...

Save and quit(:wq)

5 Execute the playbook

ansible-playbook configure\_apache2.yml -b