Ansible

A tool that helps you automate IT tasks.

* Execute tasks from your own machine remotely.
* configuration / installation / deployment steps in a single YAML file
* Re-use same file multiple times and for different environments.
* More reliable and less error prone.

**Ansible is agentless**Ansible does not need anything installed in the target machines.

Ansible works with modules. Modules are small programs that do the actual work. They get send to the control machine to the target server... do their job and then are removed.  
  
Examples of modules are:

* Start Docker Container
* Create or copy a file
* Install Nginx Server

The list of modules can be seen in the Ansible official documentation.  
  
Ansible uses simple YAML language, it is upper intuitive.

**Ansible Playbooks**  
  
Multiple modules in a certain sequence grouped together.  
  
Sequential modules are grouped in tasks. Where each task makes sure the module gets executed with certan arguments and also describes a task using a name.  
  
Example:

- hosts: databases 🡺 where should these tasks be executed  
 remote\_user: root 🡺 which user will be used  
 vars:  
 tablename: foo

tableowner: someuser

tasks:  
 - name: Rename table foo to bar  
 postgresql\_table:  
 table: {{ tablename }}

rename: bar  
  
- name: Set owner to someuser  
 postresql\_table:  
 name: {{ tablename }}

owner: {{ tableowner }}

**Task:**  
The units of action in ansible. Ansible executes each task one by one and in order from top to bottom.

**Play:**  
  
Define which tasks, in which hosts, by which users.  
The function of a play is to map a set of instructions defined against a particular host.  
  
**Playbook (The yml file):**  
  
Orchestrates the module execution. A playbook is a file.  
In a file there can be many Plays defined.  
  
The attribute “name” is used to uniquely identify its purpose.   
  
A YAML file starts with --- (3 hyphens)  
  
**Hosts (-hosts)**  
  
In Ansible there is a hosts file that keeps the inventory (all the machines involved in Ansible tasks execution). This file is a list of managed nodes.  
So this attribute is mapped to the values inside this hosts file.

The hosts file looks like:

10.24.0.100  
  
[webservers]  
10.24.0.1  
10.24.0.2  
  
[databases]  
10.24.0.7  
10.24.0.8

With Ansible we can create an alternative of Docker file … and do it in a more generic way that allows us to reproduce those steps on many environments.  
  
You can manage not only the Docker Container but the Host / Storage / Network where the Docker container is running.  
  
**UI dashboard from Red Hat**  
  
Centrally store all the automation tasks across teams .. grant permissions for those teams … manage inventory (which things have run and their status).  
  
**Advantages of Ansible over Pupped and Chef**  
1. Ansible users simple YAML, whereas the others user Ruby (you need to learn a new language).  
  
2. Ansible is Agentless (no installation on target servers is needed), whereas the others require installation on target machines (and this means a maintenance cost for updating / upgrading etc … ).

Configuration Management

It maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise’s hardware and software.

Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices.

Ansible Tower

You can centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling, integrated notifications and graphical inventory management. Easily embed Ansible Tower into existing tools and processes with REST API and CLI.

**Ansible Specific YAML Tags**

**name:**  
The name of the Ansible playbook

**hosts:**  
The lists of hosts or host group against which we want to run the task.

**vars:**  
To define the variables which you can use in your playbook.  
  
**tasks:**

All playbooks should contain tasks or a list of tasks to be executed.  
Each task internally links to a piece of code called a module. A module that should be executed, and arguments that are required for the module you want to execute.

A good practice is to give each task a name (for debugging purposes).

**block:**  
Ansible syntax to execute a given block. It is recommended that each block has a name (for debugging purposes).

**action:**   
The code next to action tag is the task to be executed.

**register:**  
The output of the action is registered using the register keyword and Output is the variable name which holds the action output.

**always:**  
it states that below will always be executed.

**msg:**  
Displays the message.

Example:

block:

- name: Install Tomcat artifacts

action: >

yum name = "demo-tomcat-1" state = present

register: Output 🡺 Registering this variable using as value the output of the action.

always:

- debug:

msg:

- "Install Tomcat artifacts task ended with message: {{Output}}"

- "Installed Tomcat artifacts - {{Output.changed}}" 🡺 Whether the output got changed

**Exception Handling in Playbooks**  
  
***rescue*** and ***always*** are the keywords specific to exception handling.  
  
***rescue*** gets executed if there is an error in the ***block*** section.  
***always*** is always executed.

Example:

tasks:

- name: Name of the task to be executed

block:

- debug: msg = 'Just a debug message , relevant for logging'

- command: <the command to execute>

rescue:

- debug: msg = 'There was an exception.. '

- command: <Rescue mechanism for the above exception occurred)

always:

- debug: msg = "this will execute in all scenarios. Always will get logged"

**Loops in Ansible**

* The loop keyword is equivalent to with\_list, and is the best choice for simple loops.
* The loop keyword will not accept a string as input

**ansible-lint**

You can use [ansible-lint](https://docs.ansible.com/ansible-lint/index.html) for detailed, Ansible-specific feedback on your playbooks before you execute them.

Each error description is described: <https://docs.ansible.com/ansible-lint/rules/default_rules.html>

$ ansible-lint verify-apache.yml

**[**403**]** Package installs should not use latest

verify-apache.yml:8

Task/Handler: ensure apache is at the latest version

Ansible Roles

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files (to allow modularity on complex playbooks).  
Each role can be reused on other playbooks.

Each role is a directory tree in itself. The role name is the directory name within the /roles directory.  
  
**Using Roles in Playbook**

---

- hosts: tomcat-node

roles:

- {role: install-tomcat}

- {role: start-tomcat}

**Build your own Inventory**

Its default location it: */etc/ansible/hosts*

You can specify a different inventory in the command line by using the *-i <path>* option.

Example of an inventory (set in the INI format):

mail.example.com

[webservers] 🡺 Group Names

foo.example.com:5309

bar.example.com

[dbservers]

one.example.com

two.example.com

three.example.com  
  
# specifying the connection, user (also port can be specified) for this host  
other2.example.com ansible\_connection=ssh ansible\_user=myotheruser

# creating the “alias” jumper

jumper ansible\_port=5555 ansible\_host=192.0.2.50

**Note:**   
If a variable value set in an INI inventory must be a certain type (for example, a string or a boolean value), always specify the type with a filter in your task. Do not rely on types set in INI inventories when consuming variables.

**Assigning a variable to a group:**

[atlanta]

host1

host2

[atlanta:vars] 🡪 “:vars”

ntp\_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

**Inheriting variable values: group variables for groups of groups  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Organizing hosts and Group variables**Ansible loads host and group variable files by searching paths relative to the inventory file or the playbook file. If your inventory file at */etc/ansible/hosts* contains a host named ‘*foosball’* that belongs to two groups, ‘*raleigh’* and ‘*webservers’*, that host will use variables in YAML files at the following locations:

[atlanta]

host1

host2

[raleigh]

host2

host3

[southeast:children]

atlanta

raleigh

[southeast:vars]

some\_server=foo.southeast.example.com

halon\_system\_timeout=30

/etc/ansible/***group\_vars***/raleigh *# can optionally end in '.yml', '.yaml', or '.json'*

/etc/ansible/***group\_vars***/webservers

/etc/ansible/***host\_vars***/foosball

**Note:** By default, those locations are a YAML file.  
But you can even create a directory instead and files within that directory or subdirectories will be loaded in lexicographical order.

You can also add *group\_vars/* and *host\_vars/* directories to your playbook directory. The ansible-playbook command looks for these directories in the current working directory by default. Other Ansible commands (for example, ansible, ansible-console, and so on) will only look for *group\_vars/* and *host\_vars/* in the inventory directory. If you want other commands to load group and host variables from a playbook directory, you must provide the *--playbook-dir* option on the command line.

**Example: Group by function**  
By using host groups, you could define firewall rules inside a playbook or role affecting only database servers:

**-** **hosts:** dbservers

**tasks:**

**-** **name:** Allow access from 10.0.0.1

**ansible.builtin.iptables:**

**chain:** INPUT

**jump:** ACCEPT

**source:** 10.0.0.1

**Default Groups**  
  
There are two default groups: ***all*** and ***ungrouped***. The ***all*** group contains every host. The ***ungrouped*** group contains all hosts that don’t have another group aside from ***all***.

**Templating (Jinja2)**  
Ansible uses Jinja2 templating to enable dynamic expressions and access to variables.

All templating happens on the Ansible controller **before** the task is sent and executed on the target machine, so this Jinja2 is only needed on the controller node.  
  
Among things we can accomplish with Jinja2 are:

* Get the current time and format a string to a date time string
* Using filters to manipulate data  
  \* Handling undefined variables  
  \* Randomizing data  
  \* Managing list variables  
  \*Manipulating text  
  \* etc…
* Tests  
  \* Testing strings:  
    
    
    
    
    
    
    
    
  \* Vault  
   You can test whether a variable is vault encrypted value using the *vault\_encrypted* test.  
    
  \* Comparing versions:

**vars:**

**my\_version:** 1.2.3

**tasks:**

**-** **debug:**

**msg:** "my\_versionishigherthan1.0.0"

**when:** my\_version is version('1.0.0', '>')

**vars:**

**url:** "http://example.com/users/foo/resources/bar"

**tasks:**

**-** **debug:**

**msg:** "matchedpattern1"

**when:** url is match("http://example.com/users/.\*/resources/")

* Lookups

**vars:**

**file\_contents:** "{{ **lookup(**'file'**,** 'path/to/file.txt'**)** }}"

Roles

Roles are to reuse content.

An Ansible role has a defined directory structure with eight main standard directories.

And you can omit any directories the role does not use.   
  
By default Ansible will look in each directory within a role for a main.yml file for relevant content .

* **tasks/main.yml**  
  The main list of tasks that the role executed
* **handlers/main.yml**  
  Handlers, which may be used within or outside this role.
* **library/my\_module.py**  
  Modules, which may be used for this role (this is called embedding modules & plugins into roles).
* **defaults/main.yml**  
  Default variables for the role. These variables have the lowest priority of any other variables available.
* **vars/main.yml**  
  Other variables of the role.
* **files/main.yml**  
  files that the role deploys
* **templates/main.yml**  
  Templates that the role deploys.
* **meta/main.yml**  
  Metadata for the role, including role dependencies.

You can also add other YAML files within some directories and include them in the other *.yml* files.  
For example, including the *redhat.yml* file in the *tasks/main.yml* file looks as follows:

*# roles/example/tasks/main.yml*

**-** **name:** Install the correct web server for RHEL

**import\_tasks:** redhat.yml

**when:** ansible\_facts['os\_family']|lower == 'redhat'

**Where are Roles located?**  
  
By default, Ansible looks for roles in two locations:

* in a directory called *roles/* relative to the playbook file
* in *etc/ansible/roles*

But if you have the roles in different locations (which is not recommended because it would be harder to share them … and the idea is to reuse them) … then you have two options:

1. set the *roles\_path* configuration option
2. You can call that role using a fully qualified path:

**---**

**-** **hosts:** webservers

**roles:**

**-** **role:** '/path/to/my/roles/common'

**Using Roles**

Roles can be used in three ways:

* At the play level with the *roles* option: This is the classic way of using roles in a play.  
    
  **Note:** Roles added in a *roles* section run before any other tasks in a playbook.  
    
    
    
    
    
    
    
  The tasks / handlers / variables / etc.. is added to the play.  
  Ansible treats the roles as **static imports** and processes them **during** playbook **parsing**.  
    
  You can also pass other keywords to the roles option:

**---**

**-** **hosts:** webservers

**roles:**

**-** common

**-** webservers

**---**

**-** **hosts:** webservers

**roles:**

**-** common

**-** **role:** foo\_app\_instance

**vars:**

**dir:** '/opt/a'

**app\_port:** 5000

**tags:** typeA

**-** **role:** foo\_app\_instance

**vars:**

**dir:** '/opt/b'

**app\_port:** 5001

**tags:** typeB

**Note:** When you add a **tag** to the role option, Ansible applies the tag to ALL tasks within the role.

**Note:** When using ***vars:*** within the ***roles:*** section of a playbook, the variables are added to the play variables, making them available to all tasks within the play.

* At the tasks level with *include\_role*: You can reuse roles dynamically anywhere in the tasks section of a play using include\_role.  
    
  **Note:** roles in this section run in the order they are defined.  
  Tasks defined previous to the task that imports the rule, run first.  
    
  Including a role:

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Print a message

**ansible.builtin.debug:**

**msg:** "thistaskrunsbeforetheexamplerole"

**-** **name:** Include the example role

**include\_role:**

**name:** example

Including a role and defining other keywords:

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Include the foo\_app\_instance role

**include\_role:**

**name:** foo\_app\_instance

**vars:**

**dir:** '/opt/a'

**app\_port:** 5000

**tags:** typeA

...

Including a Role conditionally:

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Include the some\_role role

**include\_role:**

**name:** some\_role

**when:** "ansible\_facts['os\_family']=='RedHat'"

**Note:** When you add a tag to an ***include\_role*** task, Ansible applies the tag only to the include itself. This means you can pass *--tags* to run only selected tasks from the role.

* At the tasks level with *import\_role*: You can reuse roles statically anywhere in the tasks section of a play using *import\_role*.

**Note:** The behavior is the same as using the roles keyword.

**Note:** When you add a tag to an *import\_role* statement, Ansible applies the tag to *all* tasks within the role.

**Note:** It is also possible to include keywords when importing roles.

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Import the foo\_app\_instance role

**import\_role:**

**name:** foo\_app\_instance

**vars:**

**dir:** '/opt/a'

**app\_port:** 5000

...

**Running a Role multiple times in a Playbook**  
  
Ansible only executes each role once, even if you define it multiple times, unless the parameters defined on the role are different for each definition, or if it is specified in the configuration.

**Note:** By default, Ansible runs the role only once, even if it is specified several times.  
  
There are two options to force Ansible to run a role more than once:

1. Passing different parameters

**---**

**-** **hosts:** webservers

**roles:**

**-** **role:** foo

**message:** "first"

**-** **role:** foo

**message:** "second"

1. Using *allow\_duplicates: true*  
   Adding it to the *meta/main.yml* file for the role:

*# playbook.yml*

**---**

**-** **hosts:** webservers

**roles:**

**-** foo

**-** foo

*# roles/foo/meta/main.yml*

**---**

**allow\_duplicates:** true

**Using Role dependencies**To automatically pull in other roles when using a role.  
  
**Note:** Role dependencies are not executed when included, only roles within the keyword *roles:* are executed.  
**Note:** Ansible runs the roles listed under *dependencies* first, then runs the role that lists them.Role dependencies are stored in the ***meta/main.yml*** file within the role directory.

*# roles/myapp/meta/main.yml*

**---**

**dependencies:**

**-** **role:** apache

**vars:**

**apache\_port:** 80

**-** **role:** postgres

**vars:**

**dbname:** blarg

**other\_parameter:** 12

**Embedding modules and plugins in roles**If you write a custom module or plugin, or if you overwrite any existing one … you can add those to the role:To add a module or plugin to a role:  
Add a directory named ***library*** and then include the module directly inside the ***library*** directory.

**The structure looks as follows:**

roles/

my\_custom\_modules/ 🡺 Name of the role

library/ 🡺 Custom Modules directory

module1

module2

The module will be usable in the role itself, as well as any roles that are called *after* this role.

**Embedding custom filters:**

roles/

my\_custom\_filter/ 🡺 Name of the role

filter\_plugins 🡺 Custom plugins/filters directory.

filter1

filter2

**Note:** These filters can then be used in a Jinja template in any role called after ‘my\_custom\_filter’.

**Sharing Roles: Ansible Galaxy**  
[Ansible Galaxy](https://galaxy.ansible.com/) is a free site for finding, downloading, rating, and reviewing all kinds of community-developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

The client ansible-galaxy is included in Ansible. The Galaxy client allows you to download roles from Ansible Galaxy, and also provides an excellent default framework for creating your own roles.

**Custom Module vs Action Plugin**

An action plugin may be the best way to get the functionality you want. Action plugins run on the control node instead of on the managed node, and their functionality is available to all modules.

**Tip for learning:**  
Look at people’s Ansible Galaxy roles by visiting the github repo of each role.

**Use Role with Ansible Galaxy**  
Go to: <https://galaxy.ansible.com/search>  
If Ansible is 2.9 or more, you can install the roles using ansible-galaxy .. otherwise it would need to be done manually (downloading the tar file and manually applying it to the desired location).  
  
  
**Ansible Tags**  
If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbooks.  
You can do this using Ansible Tags.

Adding tags with the tags keywords  
  
\* Adding tasks to individual tasks

**tasks:**

**-** **name:** Install the servers

**ansible.builtin.yum:**

**name:**

**-** httpd

**-** memcached

**state:** present

**tags:**

**-** packages

**-** webservers

**-** **name:** Configure the service

**ansible.builtin.template:**

**src:** templates/src.j2

**dest:** /etc/foo.conf

**tags:**

**-** configuration

- packages

\*Adding tags to includes  
**Note:** Tags on an *include\_role* (dynamic import of a role), does not apply the tag to the tags of the role.

**Note:** With dynamic re-use (**includes… example include\_tasks or include\_roles**), the tags you define apply only to the include itself. If you need tag inheritance, use a **static import**.

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Include the bar role

**include\_role:**

**name:** bar

**tags:**

**-** foo

**Tag inheritance: adding tags to multiple tasks**If you want to apply the same tag or tags to multiple tasks without adding a tags line to every task, you can define the tags at the level of your play or block, or when you add a role or import a file.

**Note:** With roles and imports, Ansible appends the tags set by the roles section or import to any tags set on individual **tasks** or **blocks** within the role or imported file.

**Adding tags to blocks**Adds the tags to all the tasks within the block

*# myrole/tasks/main.yml*

**tasks:**

**-** **name:** ntp tasks

**tags:** ntp

**block:**

**-** **name:** Install ntp

**ansible.builtin.yum:**

**name:** ntp

**state:** present

**-** **name:** Configure ntp

**ansible.builtin.template:**

**src:** ntp.conf.j2

**dest:** /etc/ntp.conf

**notify:**

**-** restart ntpd

**Adding tags to plays**If all the tasks within the play should get the same tag, you can add the tag to the level of the play.

**-** **hosts:** all

**tags:** ntp

**tasks:**

**-** **name:** Install ntp

**ansible.builtin.yum:**

**name:** ntp

**-** **name:** Enable and run ntpd

**ansible.builtin.service:**

**name:** ntpd

**state:** started

**-** **hosts:** fileservers

**tags:** filesharing

**tasks:**

...

**Adding tags to roles**There are three ways to add tags to roles:  
1. Using the ***roles*** (which applies the tags to all the roles and their tasks).

**roles:**

**-** **role:** webserver

**vars:**

**port:** 5000

**tags:** **[** **web,** **foo** **]**

2. Using ***import\_role*** in your playbook. Which add the tag to all the tasks within the role

3. Manually adding tags to the tasks within the role (which is the only way to later select certain tasks within the role based on their tags).

**---**

**-** **hosts:** webservers

**tasks:**

**-** **name:** Import the foo role

**import\_role:**

**name:** foo

**tags:**

**-** bar

**-** baz

**-** **name:** Import tasks from foo.yml

**import\_tasks:** foo.yml

**tags:** **[** **web,** **foo** **]**

**Tag inheritance for includes:  
blocks and the apply keyword.**

By default, Ansible **does not** apply tag inheritance to dynamic re-use with *include\_role* and *include\_tasks*.

If you want tag inheritance, you probably want to use imports.

However, there is an option. You can use the apply keyword:

**-** **name:** Apply the db tag to the include and to all tasks in db.yaml

**include\_tasks:**

**file:** db.yml

*# adds 'db' tag to tasks within db.yml*

**apply:**

**tags:** db

*# adds 'db' tag to this 'include\_tasks' itself*

**tags:** db

Or you can use a block:

**-** **block:**

**-** **name:** Include tasks from db.yml

**include\_tasks:** db.yml

**tags:** db

**Special tags: always and never**Ansible reserves two tag names for special behavior: ***always*** and ***never***. If you assign the always tag to a task or play, Ansible will always run that task or play, unless you specifically skip it *(--skip-tags always*).

**Note:** If you assign the *never* tag to a task or play, Ansible will skip that task or play unless you specifically request it (*--tags never*).

**Selecting or skipping tags when you run a playbook**

 If you have added a tag at the block or play level, with roles, or with an import, that tag applies to every task within the block, play, role, or imported role or file.  
  
**Command line options:**

* --tags all (runs all tasks; ignore tags. This is the default, when no options specified)
* --tags “tag1,tag2”
* --skip-tags “tag1,tag2”
* --tags tagged
* --tags untagged

**Previewing the results of using tags  
  
--list-tags 🡪 generate a list of available tags**

ansible-playbook example.yml --list-tags

**--list-tasks 🡪 when used with**

ansible-playbook example.yml --tags "configuration,packages" --list-tasks

**Example of Selectively running tagged tasks in re-usable files**mixed.yml

*# mixed.yml*

**tasks:**

**-** **name:** Run the task with no tags

**ansible.builtin.debug:**

**msg:** this task has no tags

**-** **name:** Run the tagged task

**ansible.builtin.debug:**

**msg:** this task is tagged with mytag

**tags:** mytag

**-** **block:**

**-** **name:** Run the first block task with mytag

...

**-** **name:** Run the second block task with mytag

...

**tags:**

**-** mytag

myplaybook.yml

*# myplaybook.yml*

**-** **hosts:** all

**tasks:**

**-** **name:** Run tasks from mixed.yml

**include\_tasks:**

**name:** mixed.yml

**tags:** mytag

**Note:** When you run the playbook with ***ansible-playbook  -i hosts myplaybook.yml --tags "mytag"***, Ansible skips the task with no tags, runs the tagged individual task, and runs the two tasks in the block.

**1. Develop a custom module**

[**https://docs.ansible.com/ansible/latest/dev\_guide/developing\_modules.html#developing-modules**](https://docs.ansible.com/ansible/latest/dev_guide/developing_modules.html#developing-modules)

**2. Develop a custom plugin/filter**

[**https://docs.ansible.com/ansible/latest/dev\_guide/developing\_plugins.html#developing-plugins**](https://docs.ansible.com/ansible/latest/dev_guide/developing_plugins.html#developing-plugins)