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'**)** }}"