Ansible

## Basics. What will be installed

Ansible by default manages machines over the SSH protocol.

Once Ansible is installed, it will not add a database, and there will be no daemons to start or keep running. You only need to install it on one machine (which could easily be a laptop or a Virtual machine) and it can manage an entire fleet of remote machines from that central point. When Ansible manages remote machines, it does not leave software installed or running on them.

On the managed nodes, you need a way to communicate, which is normally ssh. By default this uses sftp. If that’s not available, you can switch to scp in ansible.cfg. You also need Python 2.6 or later.

When speaking with remote machines, Ansible by default assumes you are using SSH keys. SSH keys are encouraged but password authentication can also be used where needed by supplying the option --ask-pass. If using sudo features and when sudo requires a password, also supply --ask-become-pass

## Command Line Tools

ansible

ansible-playbook

ansible-vault

ansible-inventory

….

and more

**Time for some demo here:)**

## First Commands

Time to get started with some basics.

Edit (or create) /etc/ansible/hosts and put one or more remote systems in it. Your public SSH key should be located in authorized\_keys on those systems:

192.0.2.50

aserver.example.org

bserver.example.org

This is an inventory file, which is also explained in greater depth later!

We’ll assume that we are using SSH keys for authentication.

Now ping all your nodes:

$ ansible all -m ping

Ansible will attempt to remote connect to the machines using your current user name, just like SSH would. To override the remote user name, just use the ‘-u’ parameter.

If you would like to access sudo mode, there are also flags to do that:

# as bruce, sudoing to root

$ ansible all -m ping -u bruce -b

# as bruce, sudoing to batman

$ ansible all -m ping -u bruce -b --become-user batman

Now run a live command on all of your nodes:

$ ansible all -a "/bin/echo hello"

# Inventory

Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible’s inventory. You can specify a different inventory file using the -i <path> option on the command line.

## Hosts and Groups

The inventory file can be in one of many formats, depending on the inventory plugins you have. For this example, the format INI-like (one of Ansible’s defaults) and looks like this:

mail.example.com

[webservers]

foo.example.com

bar.example.com

[dbservers]

one.example.com

two.example.com

The headings in brackets are group names, which are used in classifying systems.

A YAML version would look like:

all:

hosts:

mail.example.com

children:

webservers:

hosts:

foo.example.com:

bar.example.com:

dbservers:

hosts:

one.example.com:

two.example.com:

three.example.com:

Suppose you have just static IPs and want to set up some aliases that live in your host file. You can also describe hosts via variables:

**In INI:**

jumper ansible\_port=5555 ansible\_host=192.0.2.50

In the above example, trying to ansible against the host alias “jumper” (which may not even be a real hostname) will contact 192.0.2.50 on port 5555. Note that this is using a feature of the inventory file to define some special variables. Generally speaking, this is not the best way to define variables that describe your system policy, but we’ll share suggestions on doing this later.

Note

Values passed in the INI format using the key=value syntax are not interpreted as Python literal structure (strings, numbers, tuples, lists, dicts, booleans, None), but as a string. For example var=FALSE would create a string equal to ‘FALSE’. Do not rely on types set during definition, always make sure you specify type with a filter when needed when consuming the variable.

If you are adding a lot of hosts following similar patterns, you can do this rather than listing each hostname:

[webservers]

www[01:50].example.com

You can also select the connection type and user on a per host basis:

[targets]

localhost ansible\_connection=local

other1.example.com ansible\_connection=ssh ansible\_user=mpdehaan

other2.example.com ansible\_connection=ssh ansible\_user=mdehaan

## [Host Variables](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id7)

As described above, it is easy to assign variables to hosts that will be used later in playbooks:

[atlanta]

host1 http\_port=80 maxRequestsPerChild=808

host2 http\_port=303 maxRequestsPerChild=909

## [Group Variables](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id8)

Variables can also be applied to an entire group at once:

[atlanta]

host1

host2

[atlanta:vars]

ntp\_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

## [Groups of Groups, and Group Variables](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id9)

It is also possible to make groups of groups using the :children suffix in INI or the children: entry in YAML. You can apply variables using :vars or vars::

[atlanta]

host1

host2

[raleigh]

host2

host3

[southeast:children]

atlanta

raleigh

[southeast:vars]

some\_server=foo.southeast.example.com

halon\_system\_timeout=30

self\_destruct\_countdown=60

escape\_pods=2

[usa:children]

southeast

northeast

southwest

northwest

## [Default groups](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id10)

There are two default groups: all and ungrouped. all contains every host. ungrouped contains all hosts that don’t have another group aside from all. Every host will always belong to at least 2 groups. Though all and ungrouped are always present, they can be implicit and not appear in group listings like group\_names.

## [Splitting Out Host and Group Specific Data](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id11)

The preferred practice in Ansible is to not store variables in the main inventory file.

In addition to storing variables directly in the inventory file, host and group variables can be stored in individual files relative to the inventory file (not directory, it is always the file).

**Time to show a reference to our inventory file**

## [List of Behavioral Inventory Parameters](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id12)

Setting the following variables controls how A**nsible** interacts with remote hosts.

**General for all connections:**

ansible\_host

The name of the host to connect to, if different from the alias you wish to give to it.

ansible\_port

The ssh port number, if not 22

ansible\_user

The default ssh user name to use.

Specific to the SSH connection:

ansible\_ssh\_pass **The ssh password to use (we never store this variable in plain text; we use a vaults.)**

## [Non-SSH connection types](https://docs.ansible.com/ansible/2.4/intro_inventory.html" \l "id13)

ansible\_connection=…..

local

This connector can be used to deploy/use the playbook to the control machine itself.

docker

This connector deploys the playbook directly into Docker containers using the local Docker client. The following parameters are processed by this connector:

ansible\_host

The name of the Docker container to connect to.

ansible\_user

The user name to operate within the container. The user must exist inside the container.

ansible\_become

If set to true the become\_user will be used to operate within the container.

ansible\_docker\_extra\_args

# Playbooks

If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

Playbooks are designed to be human-readable and are developed in a basic text language. There are multiple ways to organize playbooks and the files they include.

## Creating Reusable Playbooks

While it is possible to write a playbook in one very large file (and you might start out learning playbooks this way), eventually you’ll want to reuse files and start to organize things. In Ansible, there are three ways to do this: includes, imports, and roles.

Includes and imports (added in 2.4) allow users to break up large playbooks into smaller files, which can be used across multiple parent playbooks or even multiple times within the same Playbook.

Roles allow more than just tasks to be packaged together and can include variables, handlers, or even modules and other plugins. Unlike includes and imports, roles can also be uploaded and shared via Ansible Galaxy.

## Dynamic vs. Static

Ansible has two modes of operation for reusable content: dynamic and static.

If you use any import\* Task (import\_playbook, import\_tasks, etc.), it will be static. If you use any include\* Task (include\_tasks, include\_role, etc.), it will be dynamic.

### Differences Between Static and Dynamic

The two modes of operation are pretty simple:

* Ansible pre-processes all static imports during Playbook parsing time.
* Dynamic includes are processed during runtime at the point in which that task is encountered.

When it comes to Ansible task options like tags and conditional statements (when:):

* For static imports, the parent task options will be copied to all child tasks contained within the import.
* For dynamic includes, the task options will only apply to the dynamic task as it is evaluated, and will not be copied to child tasks.

Roles are a somewhat special case in everything. The include\_role option to allow roles to be executed inline with other tasks.

### Tradeoffs Between Includes and Imports

Using include\* vs. import\* has some advantages as well as some tradeoffs which users should consider when choosing to use each:

The primary advantage of using include\* statements is looping. When a loop is used with an include, the included tasks or role will be executed once for each item in the loop.

Using include\* does have some limitations when compared to import\* statements:

* Tags which only exist inside a dynamic include will not show up in –list-tags output.
* Tasks which only exist inside a dynamic include will not show up in –list-tasks output.
* You cannot use notify to trigger a handler name which comes from inside a dynamic include (see note below).

Using import\* can also have some limitations when compared to dynamic includes:

* loops cannot be used with imports at all.
* variables from inventory sources (host/group vars, etc.) cannot be used.

**Time to show a reference to our “create\_limited\_access\_subnet.yml” file**

## **[Variables Defined in Inventory](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id16)**

We already talked about these :) So skipping…

## [Variables Defined in a Playbook](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id17)

In a playbook, it’s possible to define variables directly inline like so:

- hosts: webservers

vars:

http\_port: 80

This can be nice as it’s right there when you are reading the playbook.

## [Using Variables: About Jinja2](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id19)

Ansible allows you to reference variables in your playbooks using the Jinja2 templating system.

For instance, in a simple template, you can do something like:

My amp goes to {{ max\_amp\_value }} - basic form of variable substitution.

This is also valid directly in playbooks, and you’ll occasionally want to do things like:

template: src=foo.cfg.j2 dest={{ remote\_install\_path }}/foo.cfg

Above we used a variable to help decide where to place a file.

**Time to show: infrastructure-github/create\_perimeter\_config/prepare\_hp\_config\_file.yml + infrastructure-github/ansible\_hpe\_cw7/configs/\*\***

## [Registered Variables](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id27)

Another major use of variables is running a command and using the result of that command to save the result into a variable. Results will vary from module to module.

Here’s a quick syntax example:

- hosts: web\_servers

tasks:

- shell: /usr/bin/foo

register: foo\_result

ignore\_errors: True

- shell: /usr/bin/bar

when: foo\_result.rc == 5

## [Magic Variables](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id29)

How to access other nodes variables:

{{ hostvars['test.example.com']['ansible\_distribution'] }}

A frequently used idiom is walking a group to find all IP addresses in that group

{% for host in groups['app\_servers'] %}

{{ hostvars[host]['ansible\_eth0']['ipv4']['address'] }}

{% endfor %}

## [Variable File Separation](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id30)

It’s a great idea to keep your playbooks under source control, but you may wish to make the playbook source public while keeping certain important variables private. Similarly, sometimes you may just want to keep certain information in different files, away from the main playbook.

You can do this by using an external variables file, or files, just like this:

---

- hosts: all

remote\_user: root

vars:

favcolor: blue

vars\_files:

- /vars/external\_vars.yml

tasks:

- name: this is just a placeholder

command: /bin/echo foo

The contents of each variables file is a simple YAML dictionary

## [Variable Precedence: Where Should I Put A Variable?](https://docs.ansible.com/ansible/2.4/playbooks_variables.html" \l "id32)

In 2.x, the order of precedence is following:

* role defaults
* inventory file or script group vars
* inventory group\_vars/all
* playbook group\_vars/all
* inventory group\_vars/\*
* playbook group\_vars/\*
* inventory file or script host vars
* inventory host\_vars/\*
* playbook host\_vars/\*
* host facts
* play vars
* play vars\_prompt
* play vars\_files
* role vars (defined in role/vars/main.yml)
* block vars (only for tasks in block)
* task vars (only for the task)
* role (and include\_role) params
* include params
* include\_vars
* set\_facts / registered vars
* extra vars (always win precedence)!!!

## Loops (we will not go into details here)

**Standard Loops**   **Nested Loops**

Looping over Hashes Looping over Files

Looping over Fileglobs **Looping over File trees**

Looping over Parallel Sets of Data Looping over Subelements

Looping over Integer Sequences  **Random Choices**

**Do-Until Loops**  **Finding First Matched Files**

**Iterating Over The Results of a Program Execution**  Looping Over A List With An Index

Using ini file with a loop Flattening A List

Using register with a loop  **Looping over the inventory**

Loop Control Loops and Includes in 2.0 Writing Your Own Iterators

name: Recreate symlinks

file:

src: '{{ item.src }}'

dest: /web/{{ item.path }}

state: link

force: yes

mode: '{{ item.mode }}'

with\_filetree: web/

when: item.state == 'link'

msg: "{{ item }}"

with\_random\_choice:

- "go through the door"

- "drink from the goblet"

- "press the red button"

- "do nothing"

- shell: /usr/bin/foo

register: result

until: result.stdout.find("all systems go") != -1

retries: 5

delay: 10

# Blocks

For logical grouping of tasks and even in play error handling. Most of what you can apply to a single task can be applied at the block level:

Block example[¶](https://docs.ansible.com/ansible/2.4/playbooks_blocks.html" \l "id1)

tasks:

- name: Install Apache

block:

- yum: name={{ item }} state=installed

with\_items:

- httpd

- memcached

- template: src=templates/src.j2 dest=/etc/foo.conf

- service: name=bar state=started enabled=True

when: ansible\_distribution == 'CentOS'

become: true

become\_user: root

## Error Handling

Blocks also introduce the ability to handle errors in a way similar to exceptions in most programming languages:

tasks:

- name: Attempt and gracefull roll back demo

block:

- debug: msg='I execute normally'

- command: /bin/false

- debug: msg='I never execute, due to the above task failing'

rescue:

- debug: msg='I caught an error'

- command: /bin/false

- debug: msg='I also never execute :-('

always:

- debug: msg="this always executes"

## Modules Index

* [All Modules](https://docs.ansible.com/ansible/2.4/list_of_all_modules.html)
* [Cloud Modules](https://docs.ansible.com/ansible/2.4/list_of_cloud_modules.html)
* [Clustering Modules](https://docs.ansible.com/ansible/2.4/list_of_clustering_modules.html)
* [Commands Modules](https://docs.ansible.com/ansible/2.4/list_of_commands_modules.html)
* [Crypto Modules](https://docs.ansible.com/ansible/2.4/list_of_crypto_modules.html)
* [Database Modules](https://docs.ansible.com/ansible/2.4/list_of_database_modules.html)
* [Files Modules](https://docs.ansible.com/ansible/2.4/list_of_files_modules.html)
* [Identity Modules](https://docs.ansible.com/ansible/2.4/list_of_identity_modules.html)
* [Inventory Modules](https://docs.ansible.com/ansible/2.4/list_of_inventory_modules.html)
* [Messaging Modules](https://docs.ansible.com/ansible/2.4/list_of_messaging_modules.html)
* [Monitoring Modules](https://docs.ansible.com/ansible/2.4/list_of_monitoring_modules.html)
* [Net Tools Modules](https://docs.ansible.com/ansible/2.4/list_of_net_tools_modules.html)
* [Network Modules](https://docs.ansible.com/ansible/2.4/list_of_network_modules.html)
* [Notification Modules](https://docs.ansible.com/ansible/2.4/list_of_notification_modules.html)
* [Packaging Modules](https://docs.ansible.com/ansible/2.4/list_of_packaging_modules.html)
* [Remote Management Modules](https://docs.ansible.com/ansible/2.4/list_of_remote_management_modules.html)
* [Source Control Modules](https://docs.ansible.com/ansible/2.4/list_of_source_control_modules.html)
* [Storage Modules](https://docs.ansible.com/ansible/2.4/list_of_storage_modules.html)
* [System Modules](https://docs.ansible.com/ansible/2.4/list_of_system_modules.html)
* [Utilities Modules](https://docs.ansible.com/ansible/2.4/list_of_utilities_modules.html)
* [Web Infrastructure Modules](https://docs.ansible.com/ansible/2.4/list_of_web_infrastructure_modules.html)
* [Windows Modules](https://docs.ansible.com/ansible/2.4/list_of_windows_modules.html)

# What Kind of playbooks/modules/strategies/ansible command line tools do we use in our solution

Create VM from a VM template - proxmox\_kvm\_module

Create VM from Scratch - proxmox\_kvm\_module

Put VM in state(Started, Stopped, Absent, Present, Restarted, Current) - proxmox\_kvm\_module

Wait for machine port to become available

Dynamically add host to inventory during run

Install configuration to HP Comware7 Perimeter router

Copy content to remote host location and set access privileges for that content

Templates a file out to a remote server(Jinja2 templating language)

Shell script execution from within a Playbook whenever pure ansible modules functionality is limited

Blocks - allow logical grouping of tasks and in play error handling. Or conditional execution Most of what you can apply to a single task can be applied at the block level, which also makes it much easier to set data or directives common to the tasks.

Code re-usability using – task imports and includes(statements are pre-processed at the time playbooks are parsed | statements are processed as they encountered during the execution of the playbook)

Vaults

Idempotency - An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions.