**[General tips](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html" \l "id1)**

These concepts apply to all Ansible activities and artifacts.

[**Keep it simple**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id2)

Whenever you can, do things simply. Use advanced features only when necessary, and select the feature that best matches your use case. For example, you will probably not need vars, vars\_files, vars\_prompt and --extra-vars all at once, while also using an external inventory file. If something feels complicated, it probably is. Take the time to look for a simpler solution.

[**Use version control**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id3)

Keep your playbooks, roles, inventory, and variables files in git or another version control system and make commits to the repository when you make changes. Version control gives you an audit trail describing when and why you changed the rules that automate your infrastructure.

[**Playbook tips**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id4)

These tips help make playbooks and roles easier to read, maintain, and debug.

[**Use whitespace**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id5)

Generous use of whitespace, for example, a blank line before each block or task, makes a playbook easy to scan.

[**Always name tasks**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id6)

Task names are optional, but extremely useful. In its output, Ansible shows you the name of each task it runs. Choose names that describe what each task does and why.

[**Always mention the state**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id7)

For many modules, the ‘state’ parameter is optional. Different modules have different default settings for ‘state’, and some modules support several ‘state’ settings. Explicitly setting ‘state=present’ or ‘state=absent’ makes playbooks and roles clearer.

[**Use comments**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id8)

Even with task names and explicit state, sometimes a part of a playbook or role (or inventory/variable file) needs more explanation. Adding a comment (any line starting with ‘#’) helps others (and possibly yourself in future) understand what a play or task (or variable setting) does, how it does it, and why.

[**Inventory tips**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id9)

These tips help keep your inventory well organized.

[**Use dynamic inventory with clouds**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id10)

With cloud providers and other systems that maintain canonical lists of your infrastructure, use [dynamic inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#intro-dynamic-inventory) to retrieve those lists instead of manually updating static inventory files. With cloud resources, you can use tags to differentiate production and staging environments.

[**Group inventory by function**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id11)

A system can be in multiple groups. See [How to build your inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory) and [Patterns: targeting hosts and groups](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#intro-patterns). If you create groups named for the function of the nodes in the group, for example *webservers* or *dbservers*, your playbooks can target machines based on function. You can assign function-specific variables using the group variable system, and design Ansible roles to handle function-specific use cases. See [Roles](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#playbooks-reuse-roles).

[**Separate production and staging inventory**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id12)

You can keep your production environment separate from development, test, and staging environments by using separate inventory files or directories for each environment. This way you pick with -i what you are targeting. Keeping all your environments in one file can lead to surprises!

[**Keep vaulted variables safely visible**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id13)

You should encrypt sensitive or secret variables with Ansible Vault. However, encrypting the variable names as well as the variable values makes it hard to find the source of the values. You can keep the names of your variables accessible (by grep, for example) without exposing any secrets by adding a layer of indirection:

1. Create a group\_vars/ subdirectory named after the group.
2. Inside this subdirectory, create two files named vars and vault.
3. In the vars file, define all of the variables needed, including any sensitive ones.
4. Copy all of the sensitive variables over to the vault file and prefix these variables with vault\_.
5. Adjust the variables in the vars file to point to the matching vault\_ variables using jinja2 syntax: db\_password: {{ vault\_db\_password }}.
6. Encrypt the vault file to protect its contents.
7. Use the variable name from the vars file in your playbooks.

When running a playbook, Ansible finds the variables in the unencrypted file, which pulls the sensitive variable values from the encrypted file. There is no limit to the number of variable and vault files or their names.

[**Execution tricks**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id14)

These tips apply to using Ansible, rather than to Ansible artifacts.

[**Try it in staging first**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id15)

Testing changes in a staging environment before rolling them out in production is always a great idea. Your environments need not be the same size and you can use group variables to control the differences between those environments.

[**Update in batches**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id16)

Use the ‘serial’ keyword to control how many machines you update at once in the batch. See [Controlling where tasks run: delegation and local actions](https://docs.ansible.com/ansible/latest/user_guide/playbooks_delegation.html#playbooks-delegation).

[**Handling OS and distro differences**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html#id17)

Group variables files and the group\_by module work together to help Ansible execute across a range of operating systems and distributions that require different settings, packages, and tools. The group\_by module creates a dynamic group of hosts matching certain criteria. This group does not need to be defined in the inventory file. This approach lets you execute different tasks on different operating systems or distributions. For example:

**---**

**-** name**:** talk to all hosts just so we can learn about them

hosts**:** all

tasks**:**

**-** name**:** Classify hosts depending on their OS distribution

group\_by**:**

key**:** os\_{{ **ansible\_facts[**'distribution'**]** }}

*# now just on the CentOS hosts...*

**-** hosts**:** os\_CentOS

gather\_facts**:** False

tasks**:**

**-** *# tasks that only happen on CentOS go in this play*

The first play categorizes all systems into dynamic groups based on the operating system name. Later plays can use these groups as patterns on the hosts line. You can also add group-specific settings in group vars files. All three names must match: the name created by the group\_by task, the name of the pattern in subsequent plays, and the name of the group vars file. For example:

**---**

*# file: group\_vars/all*

asdf**:** 10

**---**

*# file: group\_vars/os\_CentOS.yml*

asdf**:** 42

In this example, CentOS machines get the value of ‘42’ for asdf, but other machines get ‘10’. This can be used not only to set variables, but also to apply certain roles to only certain systems.

You can use the same setup with include\_vars when you only need OS-specific variables, not tasks:

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

tasks**:**

**-** name**:** Set OS distribution dependent variables

include\_vars**:** "os\_{{ **ansible\_facts[**'distribution'**]** }}.yml"

**-** debug**:**

var**:** asdf

This pulls in variables from the group\_vars/os\_CentOS.yml file.

## [Using patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id2)

You use a pattern almost any time you execute an ad-hoc command or a playbook. The pattern is the only element of an [ad-hoc command](https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html#intro-adhoc) that has no flag. It is usually the second element:

ansible <pattern> -m <module\_name> -a "<module options>"

For example:

ansible webservers -m service -a "name=httpd state=restarted"

In a playbook the pattern is the content of the hosts: line for each play:

**-** name**:** <play\_name>

hosts**:** <pattern>

For example:

**-** name**:** restart webservers

hosts**:** webservers

Since you often want to run a command or playbook against multiple hosts at once, patterns often refer to inventory groups. Both the ad-hoc command and the playbook above will execute against all machines in the webservers group.

## [Common patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id3)

This table lists common patterns for targeting inventory hosts and groups.

| **Description** | **Pattern(s)** | **Targets** |
| --- | --- | --- |
| **All hosts** | all (or \*) |  |
| **One host** | host1 |  |
| **Multiple hosts** | host1:host2 (or host1,host2) |  |
| **One group** | webservers |  |
| **Multiple groups** | webservers:dbservers | all hosts in webservers plus all hosts in dbservers |
| **Excluding groups** | webservers:!atlanta | all hosts in webservers except those in atlanta |
| **Intersection of groups** | webservers:&staging | any hosts in webservers that are also in staging |

**Note**

You can use either a comma (,) or a colon (:) to separate a list of hosts. The comma is preferred when dealing with ranges and IPv6 addresses.

Once you know the basic patterns, you can combine them. This example:

webservers:dbservers:&staging:!phoenix

targets all machines in the groups ‘webservers’ and ‘dbservers’ that are also in the group ‘staging’, except any machines in the group ‘phoenix’.

You can use wildcard patterns with FQDNs or IP addresses, as long as the hosts are named in your inventory by FQDN or IP address:

192.0.\\*

\\*.example.com

\\*.com

You can mix wildcard patterns and groups at the same time:

one\*.com:dbservers

## [Limitations of patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id4)

Patterns depend on inventory. If a host or group is not listed in your inventory, you cannot use a pattern to target it. If your pattern includes an IP address or hostname that does not appear in your inventory, you will see an error like this:

[WARNING]: No inventory was parsed, only implicit localhost is available

[WARNING]: Could not match supplied host pattern, ignoring: \*.not\_in\_inventory.com

Your pattern must match your inventory syntax. If you define a host as an [alias](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#inventory-aliases):

atlanta**:**

host1**:**

http\_port**:** 80

maxRequestsPerChild**:** 808

host**:** 127.0.0.2

you must use the alias in your pattern. In the example above, you must use host1 in your pattern. If you use the IP address, you will once again get the error:

**[WARNING]:** Could not match supplied host pattern, ignoring**:** 127.0.0.2

## [Advanced pattern options](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id5)

The common patterns described above will meet most of your needs, but Ansible offers several other ways to define the hosts and groups you want to target.

### [Using variables in patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id6)

You can use variables to enable passing group specifiers via the -e argument to ansible-playbook:

webservers:!{{ **excluded** }}:&{{ **required** }}

### [Using group position in patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id7)

You can define a host or subset of hosts by its position in a group. For example, given the following group:

**[webservers]**

cobweb

webbing

weber

you can use subscripts to select individual hosts or ranges within the webservers group:

webservers[0] *# == cobweb*

webservers[-1] *# == weber*

webservers[0:2] *# == webservers[0],webservers[1]*

*# == cobweb,webbing*

webservers[1:] *# == webbing,weber*

webservers[:3] *# == cobweb,webbing,weber*

### [Using regexes in patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id8)

You can specify a pattern as a regular expression by starting the pattern with ~:

~(web|db).\*\.example\.com

## [Patterns and ansible-playbook flags](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#id9)

You can change the behavior of the patterns defined in playbooks using command-line options. For example, you can run a playbook that defines hosts: all on a single host by specifying -i 127.0.0.2, (note the trailing comma). This works even if the host you target is not defined in your inventory. You can also limit the hosts you target on a particular run with the --limit flag:

ansible-playbook site.yml --limit datacenter2

Finally, you can use --limit to read the list of hosts from a file by prefixing the file name with @:

ansible-playbook site.yml --limit @retry\_hosts.txt

If [RETRY\_FILES\_ENABLED](https://docs.ansible.com/ansible/latest/reference_appendices/config.html#retry-files-enabled) is set to True, a .retry file will be created after the ansible-playbook run containing a list of failed hosts from all plays. This file is overwritten each time ansible-playook finishes running.