Crawl Report for: https://docs.ansible.com/

Content from: https://docs.ansible.com/

Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible community documentation
Ansible offers open-source automation that is simple, flexible, and powerful.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
Ansible package docs home
Collection index
Modules and plugins index
Get started with Ansible
Understand the fundamentals of Ansible automation
Install the Ansible package
Run your first ad hoc command in a few easy steps
Users
Start writing Ansible playbooks
Learn about Ansible modules
Build inventory files to manage multiple hosts
Continue the Ansible user journey
Developers
Set up your development environment
Learn how Ansible works
Write custom modules or plugins
Continue the Ansible developer journey
Maintainers
Review community maintainer responsibilities
Understand Ansible contributor paths
Explore ways to grow community
Continue the Ansible community maintainer journey

Crawl Report for: https://docs.ansible.com/

Content from: https://docs.ansible.com/

Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible community documentation
Ansible offers open-source automation that is simple, flexible, and powerful.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
Ansible package docs home
Collection index
Modules and plugins index
Get started with Ansible
Understand the fundamentals of Ansible automation
Install the Ansible package
Run your first ad hoc command in a few easy steps
Users
Start writing Ansible playbooks
Learn about Ansible modules
Build inventory files to manage multiple hosts
Continue the Ansible user journey
Developers
Set up your development environment
Learn how Ansible works
Write custom modules or plugins
Continue the Ansible developer journey
Maintainers
Review community maintainer responsibilities
Understand Ansible contributor paths
Explore ways to grow community
Continue the Ansible community maintainer journey

Content from: https://docs.ansible.com/

Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible community documentation
Ansible offers open-source automation that is simple, flexible, and powerful.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
Ansible package docs home
Collection index
Modules and plugins index
Get started with Ansible
Understand the fundamentals of Ansible automation
Install the Ansible package
Run your first ad hoc command in a few easy steps
Users
Start writing Ansible playbooks
Learn about Ansible modules
Build inventory files to manage multiple hosts
Continue the Ansible user journey
Developers
Set up your development environment
Learn how Ansible works
Write custom modules or plugins
Continue the Ansible developer journey
Maintainers
Review community maintainer responsibilities
Understand Ansible contributor paths
Explore ways to grow community
Continue the Ansible community maintainer journey

Content from: https://docs.ansible.com/ansible-core/2.11_ja/index.html

Ansible Core ?????? ? Ansible Core Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.11_ja
??????????????????
?????????
Ansible Core ?????
Ansible Core ???
???????
Ansible Core ????
Ansible ??????????
Ansible ???
??????
???????
????????????
??????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible
Ansible Core ??????
GitHub ???
You are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible documentation
.
Ansible Core ??????
?
ansible-core ???
?
Ansible ? IT ??? IT ??? (????????????????????????????????)

??????????????????
Ansible ?????
ansible-core
? Ansible ??????????????????????????????????????
???????????
ansible-playbook
?
ansible-doc
??? CLI ???
YAML ????? Ansible Playbook ??????????????????conditional?block?include?include????? Ansible ??????????????
Ansible ????????????????????????????????
Ansible ??? OpenSSH ??? (??????????????????????)????????????????????????????????????
??IT ????????????????????????????????????Ansible ??
????
AnsibleFest
(Red Hat ??????Ansible ????????????????????????????????) ???????????AnsibleFest ??
Ansible
??OpenSSH ???????????????????????????????? 1 ????????????????????????????????Ansible ????????????? OS ??Ansible ??Kerberos?LDAP????????????????????????????????
??????????????????
ansible-core
?????????????Red Hat ???????????
ansible-core
????????????????????????????????? Ansible ??? Ansible ??????????????????
ansible-core
????? 2 ??? 2.10 ??
??????????????????????
?????????
Ansible ???????
Ansible ???
Ansible Core ?????
Ansible-core 2.11 ?????
Ansible-base 2.10 ?????
Ansible Core ???
???????
????
??????????? Playbook ???
??????????
???????
Playbook ???
??????????
?????
Ansible Core ????
Ansible ??????????
????
????????
Ansible ?????????

????: 11? 11, 2022

Content from: https://docs.ansible.com/ansible-core/2.12_ja/index.html

Ansible Core ?????? ? Ansible Core Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.12_ja
2.14_ja
2.13_ja
2.12_ja
??????????????????
?????????
Ansible Core ?????
Ansible Core ???
???????
Ansible Core ????
Ansible ??????????
Ansible ???
??????
???????
Collection Index
??????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible
Ansible Core ??????
GitHub ???
Ansible Core ??????
?
ansible-core ???
?
Ansible ? IT ??? IT ??? (????????????????????????????)
??????????????????

Ansible ?????
ansible-core
? Ansible ???????????????????????????????????????
????????????
ansible-playbook
?
ansible-doc
??? CLI ???
YAML ????? Ansible Playbook ???????????????????conditional?block?include?include????? Ansible ????????????
Ansible ??
Ansible ?????????????????????????????????
Ansible ??? OpenSSH ??? (??????????????????????)????????????????????????????????????
???IT ?????????????????????????????????????Ansible ??
????
AnsibleFest
(Red Hat ??????Ansible ????????????????????????????????) ???????????AnsibleFest ???
Ansible
???OpenSSH ???????????????????????????? 1 ?????????????????????????????Ansible ????????????? OS ?????????????????????????????????????Ansible ??Kerberos?LDAP??????????????????????????????
???????????????????????
ansible-core
????????????????Red Hat ???????????
ansible-core
??????????????????????????????????? Ansible ??? Ansible ????????????????
ansible-core
????? 2 ??? 2.10 ???
????????????????????
?????????
Ansible ???????
Ansible ???
Ansible Core ?????
Ansible-core 2.12 ?????
Ansible-core 2.11 ?????
Ansible-base 2.10 ?????
Ansible Core ???
???????
????
??????????? Playbook ???
??????????
???????
Playbook ???
??????????
?????
Ansible Core ????
Ansible ??????????
????
????????
Ansible ?????????

?????
Ansible ???
??????
??????????????????
???????????
Ansible ????????
??? Ansible ???????????????
??????????????
Ansible ??? Python 3
??????????
????????????????
Windows ??????????????
Cisco ACI ????????
???????????
Ansible ????
Ansible ??????????????????????
????????
????????????
ansible-core
???
Ansible ??????????????
Python API
?????????
??????????????????????
?????????
?????Galaxy ???????????????
?????? Galaxy ???????
Ansible ????????
???????
Collection Index
??????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
??????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ??????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

????: 11? 09, 2022

Content from: https://docs.ansible.com/ansible-core/2.13_ja/index.html

Ansible Core ?????? ? Ansible Core Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.13_ja
Select version:
2.15_ja
2.14_ja
2.13_ja
Search docs:
Ansible ???????
Ansible ????????
??????????????????
?????????
Ansible Core ?????
Ansible Core ???
???????
Ansible Core ????
Ansible ??????????
ansible-core ????????????
???????????????
Ansible ??????????????
Ansible ???
??????
Ansible Galaxy
Galaxy ???????
Galaxy ??????
???????
Collection Index
?????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????

ansible-core ??????
Ansible
Ansible Core ??????
Ansible Core ??????
?
ansible-core ???
?
Ansible ? IT ??? IT ??? (????????????????????????????????)
??????????????????????
Ansible ?????
ansible-core
? Ansible ??????????????????????????????????????
????????????
ansible-playbook
?
ansible-doc
??? CLI ???
YAML ????? Ansible Playbook ??????????????????conditional?block?include?include????? Ansible ???????????
Ansible ??
Ansible ????????????????????????????????
Ansible ??? OpenSSH ???
(????????????????????????)??
??IT ????????????????????????????????????Ansible
??
????
AnsibleFest
(Red Hat ??????Ansible ????????????????????????????????) ???????????AnsibleFest
??
Ansible
??OpenS
SH ???????????????????????????????? 1 ????????????????????????????????Ansible ????????????? OS
??Ansible ??Kerberos?LDAP????????????????????????????????
????????????????????????????
ansible-core
??????????????????Red Hat ???????????
ansible-core
????????????????????????????????????? Ansible ??? Ansible
????????????????????
ansible-core
????? 2 ??? 2.10
??
Ansible ???????
Ansible ????????
???????????
Playbook ???
??????????????????????
?????????
Ansible ???????
????????????????????? Ansible ???????
Ansible ???
Ansible Core ?????
Ansible-core 2.13 ?????
Ansible-core 2.12 ?????
Ansible-core 2.11 ?????

Ansible-base 2.10 ?????
Ansible Core ???
???????
??????????? Playbook ???
??????????
???????
Playbook ???
??????????
??
Ansible Core ????
Ansible ??????????
????
???????????
ansible-core ????????????
?????????????
Ansible ??????????
Ansible ?????????
??????????????????
??????????
?????
Playbook ???????????
???????
Ansible ?????????
??????????????????
??????????????????
??????????????????
GitHub ???
Ansible ??????????????
?????????
reStructuredText ???????
??????????????????
????
Ansible ???
??????
?????????????????????
???????????
????????
??? Ansible ???????????????????
??????????????????
Ansible ??? Python 3
??????????
??????????????????
Windows ??????????????????
Cisco ACI ????????
???????????
Ansible ????
Ansible ?????????????????????????
????????
????????????
ansible-core
???
Ansible ??????????????????
Python API
?????????

??????????????????
????????
?????Galaxy ???????????????
?????? Galaxy ???????
Ansible ????????
Ansible Galaxy
Galaxy ???????
Galaxy ??????????
?????????????
Galaxy ????????
Galaxy ?????????????
Galaxy ??????
Galaxy ??????????
Galaxy ???????
???????
Collection Index
??????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible-core 2.13
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

????: 7? 19, 2023

Content from: https://docs.ansible.com/ansible-core/2.14_ja/index.html

Ansible Core ?????? ? Ansible Core Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.14_ja
2.15_ja
2.14_ja
2.13_ja
Ansible ???????
MySQL ????????
??????????????????
?????????
Ansible Core ?????
Ansible Core ???
Ansible ??????????
Ansible ?????????????
Ansible Playbook ???
Ansible Vault ?????????????
Ansible ???????????????
Ansible ?????????
Ansible ? Windows ??? BSD ????????
Ansible ???????
Ansible Core ????
Ansible ??????????
ansible-core ???????????
???????????????
Ansible ??????????????
Ansible ???
??????
Ansible Galaxy
Galaxy ???????
Galaxy ??????
???????
Collection Index
??????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
??????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????

?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible
Ansible Core ??????
Ansible Core ??????
?
ansible-core ???
?
Ansible ? IT ??? IT ??? (????????????????????????????????)
??????????????????
Ansible ?????
ansible-core
? Ansible ??????????????????????????????????
????????????
ansible-playbook
?
ansible-doc
??? CLI ???
YAML ????? Ansible Playbook ??????????????????conditional?block?include?include????? Ansible ????????????
Ansible ??
Ansible ????????????????????????????????
Ansible ??? OpenSSH ???
(??????????????????????)??
??IT ????????????????????????????????????Ansible
??
????
AnsibleFest
(Red Hat ??????Ansible ????????????????????????????????) ??????????AnsibleFest
??
Ansible
??OpenS
SH ???????????????????????????????? 1 ????????????????????????????????Ansible ????????????? OS
??Ansible ??Kerberos?LDAP????????????????????????????????
????????????????????????
ansible-core
??????????????????Red Hat ??????????
ansible-core
????????????????????????????????????? Ansible ??? Ansible
????????????????????
ansible-core
????? 2 ??? 2.10
??
Ansible ???????
MySQL ????????
???????????
Playbook ???
Ansible ???
????????????????????
?????????
Ansible ???????

???????
Playbook ????
???????????
??????
Ansible ????
Ansible Core ????
Ansible ??????????
????
???????????
ansible-core ????????????
?????????????
Ansible ??????????
Ansible ?????????
??????????????
??????????
?????
Playbook ???????????
???????
Ansible ?????????
??????????????
??????????????
???????????????????
GitHub ???
Ansible ??????????????
?????????
reStructuredText ???????
??????????????
????
Ansible ???
??????
????????????????????????
???????????
????????
??? Ansible ???????????????
??????????????
Ansible ??? Python 3
??????????
???????????????????
?? YAML ??????????
Windows ??????????????
???????????
Ansible ????
Ansible ????????????????????????
????????
????????????
ansible-core
???
Ansible ??????????????
Python API
?????????
????????????????????????
?????????
?????Galaxy ???????????????????
?????? Galaxy ???????

????: 7? 19, 2023

Content from: https://docs.ansible.com/ansible-core/2.15_ja/index.html

Ansible Core ?????? ? Ansible Core Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.15_ja
Select version:
2.15_ja
2.14_ja
2.13_ja
Search docs:
Ansible getting started
MySQL ????????
??????????????????
?????????
Ansible Core ?????
Ansible Core ???
Ansible ??????????
Ansible ?????????????
Ansible Playbook ???
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Ansible ?????????
Ansible ? Windows ??? BSD ????????
Ansible tips and tricks
Ansible Core ????
Ansible ??????????
ansible-core ????????????
???????????????
Ansible ??????????????
Ansible ???
??????
Ansible Galaxy
Galaxy ???????
Galaxy ??????
???????
Collection Index
?????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)

???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible
Ansible Core ??????
Ansible Core ??????
?
ansible-core ???
?
Ansible ? IT ??? IT ??? (??????????????????????????????) ????????????????????
Ansible core, or
ansible-core
is the main building block and architecture for Ansible, and includes:
CLI tools such as
ansible-playbook
,
ansible-doc
. and others for driving and interacting with automation.
The Ansible language that uses YAML to create a set of rules for developing Ansible Playbooks and includes functions such as conditionals, blocks, includes, loops, and other Ansible imperatives.
An architectural framework that allows extensions through Ansible collections.
Ansible ??? OpenSSH ??? (???????????????????????)???????????????????????????????????????
???IT ???????????????????????????????????Ansible ??
????
AnsibleFest
(Red Hat ??????Ansible ????????????????????????????????) ???????????AnsibleFest ??
Ansible
???OpenSSH ???????????????????????????? 1 ?????????????????????????????Ansible ????????????? OS ????????????????????????????????????Ansible ??Kerberos?LDAP??????????????????????????????
This documentation covers the version of
ansible-core
noted in the upper left corner of this page. We maintain multiple versions of
ansible-core
and of the documentation, so please be sure you are using the version of the documentation that covers the version of Ansible you're using. For recent features, we note the version of Ansible where the feature was added.
ansible-core
????? 2 ?? 2.10 ??
Ansible getting started
MySQL ????????
???????????
Playbook ???
Ansible ???
????????????????????

Ansible Galaxy
Galaxy ???????
Galaxy ??????????
?????????????
Galaxy ????????
Galaxy ?????????????
Galaxy ??????
Galaxy ??????????
Galaxy ???????
???????
Collection Index
??????????????????????????
Playbook ?????
???
Ansible ????
Ansible ??????: ????????
YAML ??
Python 3 ????
???????????
?????????????
??????????
??????
?????? (FAQ)
???
Ansible ????: ?????????????
?????
Red Hat Ansible Automation Platform
Ansible Automation Hub
Ansible ???????
??????
ansible-core ??????
Ansible-core 2.15
Ansible-core 2.14
Ansible-core 2.13
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

????: 7? 25, 2023

Content from: https://docs.ansible.com/ansible-core/devel/dev_guide/testing_running_locally.html

Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
ansible-core Roadmaps
Ansible Core
Testing Ansible and Collections
Edit on GitHub
Testing Ansible and Collections
?
This document describes how to run tests using
ansible-test
.
Setup
Testing an Ansible Collection
Testing
ansible-core
Commands
Environments
Containers
Choosing a container
Custom containers
Docker and SELinux
Docker Desktop with WSL2
Configuration requirements
Setup instructions
Linux cgroup configuration
Podman
Remote virtual machines
Python virtual environments
Composite environment arguments
Additional Requirements
Environment variables
Interactive shell
Code coverage
Setup
?
Before running
ansible-test
, set up your environment for
Testing an Ansible Collection
or
Testing ansible-core
, depending on which scenario applies to you.
Warning
If you use
git
for version control, make sure the files you are working with are not ignored by
git
.
If they are,
ansible-test
will ignore them as well.

Testing an Ansible Collection
?
If you are testing an Ansible Collection, you need a copy of the collection, preferably a Git clone.
For example, to work with the
community.windows
collection, follow these steps:
Clone the collection you want to test into a valid collection root:
git
clone
https://github.com/ansible-collections/community.windows
~/dev/ansible_collections/community/windows
Important
The path must end with
/ansible_collections/{collection_namespace}/{collection_name}
where
{collection_namespace}
is the namespace of the collection and
{collection_name}
is the collection name.
Clone any collections on which the collection depends:
git
clone
https://github.com/ansible-collections/ansible.windows
~/dev/ansible_collections/ansible/windows
Important
If your collection has any dependencies on other collections, they must be in the same collection root, since
ansible-test
will not use your configured collection roots (or other Ansible configuration).
Note
See the collection?s
galaxy.yml
for a list of possible dependencies.
Switch to the directory where the collection to test resides:
cd
~/dev/ansible_collections/community/windows
Testing
ansible-core
?
If you are testing
ansible-core
itself, you need a copy of the
ansible-core
source code, preferably a Git clone.
Having an installed copy of
ansible-core
is not sufficient or required.
For example, to work with the
ansible-core
source cloned from GitHub, follow these steps:
Clone the
ansible-core
repository:
git
clone

https://github.com/ansible/ansible
~/dev/ansible
Switch to the directory where the
ansible-core
source resides:
cd
~/dev/ansible
Add
ansible-core
programs to your
PATH
:
source
hacking/env-setup
Note
You can skip this step if you only need to run
ansible-test
, and not other
ansible-core
programs.
In that case, simply run
bin/ansible-test
from the root of the
ansible-core
source.
Caution
If you have an installed version of
ansible-core
and are trying to run
ansible-test
from your
PATH
,
make sure the program found by your shell is the one from the
ansible-core
source:
which
ansible-test
Commands
?
The most commonly used test commands are:
ansible-test
sanity
- Run sanity tests (mostly linters and static analysis).
ansible-test
integration
- Run integration tests.
ansible-test
units
- Run unit tests.
Run
ansible-test
--help
to see a complete list of available commands.

Note
For detailed help on a specific command, add the
--help
option after the command.
Environments
?
Most
ansible-test
commands support running in one or more isolated test environments to simplify testing.
Containers
?
Containers are recommended for running sanity, unit and integration tests, since they provide consistent environments.
Unit tests will be run with network isolation, which avoids unintentional dependencies on network resources.
The
--docker
option runs tests in a container using either Docker or Podman.
Note
If both Docker and Podman are installed, Docker will be used.
To override this, set the environment variable
ANSIBLE_TEST_PREFER_PODMAN
to any non-empty value.
Choosing a container
?
Without an additional argument, the
--docker
option uses the
default
container.
To use another container, specify it immediately after the
--docker
option.
Note
The
default
container is recommended for all sanity and unit tests.
To see the list of supported containers, use the
--help
option with the
ansible-test
command you want to use.
Note
The list of available containers is dependent on the
ansible-test
command you are using.
You can also specify your own container.
When doing so, you will need to indicate the Python version in the container with the
--python
option.
Custom containers
?
When building custom containers, keep in mind the following requirements:
The
USER
should be

root
.
Use an
init
process, such as
systemd
.
Include
sshd
and accept connections on the default port of
22
.
Include a POSIX compatible
sh
shell which can be found on
PATH
.
Include a
sleep
utility which runs as a subprocess.
Include a supported version of Python.
Avoid using the
VOLUME
statement.
Docker and SELinux
?
Using Docker on a host with SELinux may require setting the system in permissive mode.
Consider using Podman instead.
Docker Desktop with WSL2
?
These instructions explain how to use
ansible-test
with WSL2 and Docker Desktop
without
systemd
support.
Note
If your WSL2 environment includes
systemd
support, these steps are not required.
Configuration requirements
?
Open Docker Desktop and go to the
Settings
screen.
On the the
General
tab:
Uncheck the
Start Docker Desktop when you log in
checkbox.
Check the
Use the WSL 2 based engine
checkbox.

On the
Resources
tab under the
WSL Integration
section:
Enable distros you want to use under the
Enable integration with additional distros
section.
Click
Apply and restart
if changes were made.
Setup instructions
?
Note
If all WSL instances have been stopped, these changes will need to be re-applied.
Verify Docker Desktop is properly configured (see
Configuration requirements
).
Quit Docker Desktop if it is running:
Right click the
Docker Desktop
taskbar icon.
Click the
Quit Docker Desktop
option.
Stop any running WSL instances with the command:
wsl
--shutdown
Verify all WSL instances have stopped with the command:
wsl
-l
-v
Start a WSL instance and perform the following steps as
root
:
Verify the
systemd
subsystem is not registered:
Check for the
systemd
cgroup hierarchy with the following command:
grep
systemd
/proc/self/cgroup
If any matches are found, re-check the
Configuration requirements
and follow the
Setup instructions
again.
Mount the
systemd
cgroup hierarchy with the following commands:
mkdir
/sys/fs/cgroup/systemd

mount
cgroup
-t
cgroup
/sys/fs/cgroup/systemd
-o
none,name
=
systemd,xattr
Start Docker Desktop.
You should now be able to use
ansible-test
with the
--docker
option.
Linux cgroup configuration
?
Note
These changes will need to be re-applied each time the container host is booted.
For certain container hosts and container combinations, additional setup on the container host may be required.
In these situations
ansible-test
will report an error and provide additional instructions to run as
root
:
mkdir
/sys/fs/cgroup/systemd
mount
cgroup
-t
cgroup
/sys/fs/cgroup/systemd
-o
none,name
=
systemd,xattr
If you are using rootless Podman, an additional command must be run, also as
root
.
Make sure to substitute your user and group for
{user}
and
{group}
respectively:
chown
-R
{
user
}
:
{
group
}
/sys/fs/cgroup/systemd

Podman
?
When using Podman, you may need to stop existing Podman processes after following the
Linux cgroup configuration
instructions. Otherwise Podman may be unable to see the new mount point.
You can check to see if Podman is running by looking for
podman
and
catatonit
processes.
Remote virtual machines
?
Remote virtual machines are recommended for running integration tests not suitable for execution in containers.
The
--remote
option runs tests in a cloud hosted ephemeral virtual machine.
Note
An API key is required to use this feature, unless running under an approved Azure Pipelines organization.
To see the list of supported systems, use the
--help
option with the
ansible-test
command you want to use.
Note
The list of available systems is dependent on the
ansible-test
command you are using.
Python virtual environments
?
Python virtual environments provide a simple way to achieve isolation from the system and user Python environments.
They are recommended for unit and integration tests when the
--docker
and
--remote
options cannot be used.
The
--venv
option runs tests in a virtual environment managed by
ansible-test
.
Requirements are automatically installed before tests are run.
Composite environment arguments
?
The environment arguments covered in this document are sufficient for most use cases.
However, some scenarios may require the additional flexibility offered by composite environment arguments.
The
--controller
and
--target
options are alternatives to the
--docker
,
--remote
and

--venv
options.
Note
When using the
shell
command, the
--target
option is replaced by three platform specific options.
Add the
--help
option to your
ansible-test
command to learn more about the composite environment arguments.
Additional Requirements
?
Some
ansible-test
commands have additional requirements.
You can use the
--requirements
option to automatically install them.
Note
When using a test environment managed by
ansible-test
the
--requirements
option is usually unnecessary.
Environment variables
?
When using environment variables to manipulate tests there some limitations to keep in mind. Environment variables are:
Not propagated from the host to the test environment when using the
--docker
or
--remote
options.
Not exposed to the test environment unless enabled in
test/lib/ansible_test/_internal/util.py
in the
common_environment
function.
Example:
ANSIBLE_KEEP_REMOTE_FILES=1
can be set when running
ansible-test
integration
--venv
. However, using the
--docker
option would
require running
ansible-test
shell
to gain access to the Docker environment. Once at the shell prompt, the environment variable could be set

and the tests executed. This is useful for debugging tests inside a container by following the
Debugging modules
instructions.
Interactive shell
?
Use the
ansible-test
shell
command to get an interactive shell in the same environment used to run tests. Examples:
ansible-test
shell
--docker
- Open a shell in the default docker container.
ansible-test
shell
--venv
--python
3.10
- Open a shell in a Python 3.10 virtual environment.
Code coverage
?
Code coverage reports make it easy to identify untested code for which more tests should
be written. Online reports are available but only cover the
devel
branch (see
Testing Ansible
). For new code local reports are needed.
Add the
--coverage
option to any test command to collect code coverage data. If you
aren?t using the
--venv
or
--docker
options which create an isolated python
environment then you may have to use the
--requirements
option to ensure that the
correct version of the coverage module is installed:
ansible-test
coverage
erase
ansible-test
units
--coverage
apt
ansible-test
integration
--coverage
aws_lambda
ansible-test
coverage
html
Reports can be generated in several different formats:

ansible-test
coverage
report
- Console report.
ansible-test
coverage
html
- HTML report.
ansible-test
coverage
xml
- XML report.
To clear data between test runs, use the
ansible-test
coverage
erase
command.

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible-core/devel/index.html

Ansible Core Documentation ? Ansible Core Documentation
Blog
Ansible community forum
Documentation
Ansible Core Documentation
Ansible Core
Select version:
2.19
2.18
2.17
devel
Search docs:
Ansible getting started
Getting started with Ansible
Installation, Upgrade & Configuration
Installation Guide
Ansible Core Porting Guides
Using Ansible Core
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible Core
Ansible Community Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities

Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
ansible-core Roadmaps
Ansible Core
Ansible Core Documentation
Ansible Core Documentation
?
Ansible Core, or
ansible-core
is the main building block and architecture for Ansible, and includes:
CLI tools such as
ansible-playbook
,
ansible-doc
. and others for driving and interacting with automation.
The Ansible language that uses YAML to create a set of rules for developing Ansible Playbooks and includes functions such as conditionals, blocks, includes, loops, and other Ansible imperatives.
An architectural framework that allows extensions through Ansible collections.
This documentation covers the version of
ansible-core
noted in the upper left corner of this page.
We maintain multiple versions of
ansible-core
and of the documentation, so please be sure you are using the version of the documentation that covers the version of Ansible you?re using.
For recent features, we note the version of Ansible where the feature was added.
ansible-core
releases a new major release approximately twice a year.
The core application evolves somewhat conservatively, valuing simplicity in language design and setup.
Contributors develop and change modules and plugins, hosted in collections, much more quickly.
Ansible getting started
Getting started with Ansible
Introduction to Ansible
Start automating with Ansible
Building an inventory
Creating a playbook
Ansible concepts
Installation, Upgrade & Configuration
Installation Guide
Installing Ansible
Installing Ansible on specific operating systems
Configuring Ansible
Ansible Core Porting Guides
Ansible-core 2.19 Porting Guide
Ansible-core 2.18 Porting Guide
Ansible-core 2.17 Porting Guide
Ansible-core 2.16 Porting Guide
Ansible-core 2.15 Porting Guide
Ansible-core 2.14 Porting Guide
Ansible-core 2.13 Porting Guide
Ansible-core 2.12 Porting Guide

Contributor path
ansible-core Contributors Guide
Reporting bugs and requesting features
Contributing to the Ansible Documentation
The Ansible Development Cycle
Other Tools and Programs
Working with the Ansible repo
Advanced Contributor Guide
Committers Guidelines
Release Manager Guidelines
GitHub Admins
Ansible Ecosystem Project Development Resources
Ansible documentation style guide
Linguistic guidelines
reStructuredText guidelines
Markdown guidelines
Accessibility guidelines
More resources
Extending Ansible
Developer Guide
Adding modules and plugins locally
Should you develop a module?
Developing modules
Contributing your module to an existing Ansible collection
Conventions, tips, and pitfalls
Ansible and Python 3
Debugging modules
Module format and documentation
Ansible markup
Adjacent YAML documentation files
Windows module development walkthrough
Creating a new collection
Testing Ansible
The lifecycle of an Ansible module or plugin
Developing plugins
Developing dynamic inventory
Developing
ansible-core
Ansible module architecture
Python API
Rebasing a pull request
Using and developing module utilities
Ansible collection creator path
Developing collections
Migrating Roles to Roles in Collections on Galaxy
Collection Galaxy metadata structure
Ansible architecture
Ansible Galaxy
Galaxy User Guide
Finding collections on Galaxy
Finding roles on Galaxy
Installing roles from Galaxy
Galaxy Developer Guide
Creating collections for Galaxy

Creating roles for Galaxy
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
ansible-core Roadmaps
Ansible-core 2.20
Ansible-core 2.19
Ansible-core 2.18
Ansible-core 2.17
Ansible-core 2.16
Ansible-core 2.15
Ansible-core 2.14
Ansible-core 2.13
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible-prior-versions.html

Ansible community documentation archive | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible community documentation archive
Archive page for the Ansible community documentation that provides older versions of content.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Ansible community documentation for version 2.7
Visit the documentation
Ansible community documentation for version 2.6
Visit the documentation
Ansible community documentation for version 2.5
Visit the documentation
Ansible community documentation for version 2.4
Visit the documentation
Ansible community documentation for version 2.3
Visit the documentation

Content from: https://docs.ansible.com/ansible/2.9/plugins/inventory/virtualbox.html

virtualbox ? virtualbox inventory source ? Ansible Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.9
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
User Guide
Ansible Quickstart Guide
Ansible concepts
Getting Started
How to build your inventory
Working with dynamic inventory
Patterns: targeting hosts and groups
Introduction to ad-hoc commands
Connection methods and details
Working with command line tools
Working With Playbooks
Understanding privilege escalation: become
Ansible Vault
Working With Modules
Working With Plugins
Action Plugins
Become Plugins
Cache Plugins
Callback Plugins
Cliconf Plugins
Connection Plugins
Httpapi Plugins
Inventory Plugins
Enabling inventory plugins
Using inventory plugins
Plugin List
Lookup Plugins
Netconf Plugins
Shell Plugins
Strategy Plugins
Vars Plugins
Filters
Tests
Plugin Filter Configuration
Ansible and BSD
Windows Guides
Using collections
Contributing to Ansible
Ansible Community Guide

virtualbox ? virtualbox inventory source ¶

Synopsis
Parameters
Examples
Status
Synopsis
¶
Get inventory hosts from the local virtualbox installation.
Uses a YAML configuration file that ends with virtualbox.(yml|yaml) or vbox.(yml|yaml).
The inventory_hostname is always the ?Name? of the virtualbox instance.
Parameters
¶
Parameter
Choices/
Defaults
Configuration
Comments
cache
boolean
Choices:
no
?
yes
ini entries:
[inventory]
cache = no
env:ANSIBLE_INVENTORY_CACHE
Toggle to enable/disable the caching of the inventory's source data, requires a cache plugin setup to work.
cache_connection
string
ini entries:
[defaults]
fact_caching_connection = VALUE
[inventory]
cache_connection = VALUE
env:ANSIBLE_CACHE_PLUGIN_CONNECTION
env:ANSIBLE_INVENTORY_CACHE_CONNECTION
Cache connection data or path, read cache plugin documentation for specifics.
cache_plugin
string
Default:
"memory"
ini entries:
[defaults]
fact_caching = memory
[inventory]
cache_plugin = memory
env:ANSIBLE_CACHE_PLUGIN
env:ANSIBLE_INVENTORY_CACHE_PLUGIN
Cache plugin to use for the inventory's source data.
cache_prefix
-
Default:
"ansible_inventory_"
ini entries:

[default]
fact_caching_prefix = ansible_inventory_
[inventory]
cache_prefix = ansible_inventory_
env:ANSIBLE_CACHE_PLUGIN_PREFIX
env:ANSIBLE_INVENTORY_CACHE_PLUGIN_PREFIX
Prefix to use for cache plugin files/tables
cache_timeout
integer
Default:
3600
ini entries:
[defaults]
fact_caching_timeout = 3600
[inventory]
cache_timeout = 3600
env:ANSIBLE_CACHE_PLUGIN_TIMEOUT
env:ANSIBLE_INVENTORY_CACHE_TIMEOUT
Cache duration in seconds
compose
dictionary
Default:
{}
Create vars from jinja2 expressions.
groups
dictionary
Default:
{}
Add hosts to group based on Jinja2 conditionals.
keyed_groups
list
Default:
[]
Add hosts to group based on the values of a variable.
network_info_path
-
Default:
"/VirtualBox/GuestInfo/Net/0/V4/IP"
property path to query for network information (ansible_host)
plugin
-
/
required
Choices:
virtualbox
token that ensures this is a source file for the 'virtualbox' plugin
query
dictionary
Default:
{}
create vars from virtualbox properties
running_only
boolean
Default:

"no"
toggles showing all vms vs only those currently running
settings_password_file
-
provide a file containing the settings password (equivalent to --settingspwfile)
strict
boolean
Choices:
no
?
yes
If
yes
make invalid entries a fatal error, otherwise skip and continue.
Since it is possible to use facts in the expressions they might not always be available and we ignore those errors by default.
Examples
¶
file must be named vbox.yaml or vbox.yml
simple_config_file
:
plugin
:
virtualbox
settings_password_file
:
/etc/virtulbox/secrets
query
:
logged_in_users
:
/VirtualBox/GuestInfo/OS/LoggedInUsersList
compose
:
ansible_connection
:
('indows' in vbox_Guest_OS)|ternary('winrm', 'ssh')
add hosts (all match with minishift vm) to the group container if any of the vms are in ansible_inventory'
plugin
:
virtualbox
groups
:
container
:
"'minis'
in
(inventory_hostname)"
Status
¶
This inventory is not guaranteed to have a backwards compatible interface.
[preview]
This inventory is
maintained by the Ansible Community

.
[community]
Authors
¶
UNKNOWN
Hint
If you notice any issues in this documentation, you can
edit this document
to improve it.
Hint
Configuration entries for each entry type have a low to high priority order. For example, a variable that is lower in the list will override a variable that is higher up.
Next
Previous

Last updated on May 27, 2022.

Content from: https://docs.ansible.com/ansible/2.9/reference_appendices/module_utils.html

Ansible Reference: Module Utilities ? Ansible Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.9
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
User Guide
Contributing to Ansible
Ansible Community Guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Public Cloud Guides
Network Technology Guides
Virtualization and Containerization Guides
Ansible for Network Automation
Ansible for Network Automation
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Module Index
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Release and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
AnsibleModule
Basic
Special Variables
Red Hat Ansible Tower
Logging Ansible output
Roadmaps
Ansible Roadmap
Ansible
Docs
»

Ansible Reference: Module Utilities
Edit on GitHub
For community users, you are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible community documentation
. For Red Hat customers, see
the Red Hat AAP platform lifecycle
.
Ansible Reference: Module Utilities
¶
This page documents utilities intended to be helpful when writing
Ansible modules in Python.
AnsibleModule
¶
To use this functionality, include
from
ansible.module_utils.basic
import
AnsibleModule
in your module.
class
ansible.module_utils.basic.
AnsibleModule
(
argument_spec
,
bypass_checks
=
False
,
no_log
=
False
,
check_invalid_arguments
=
None
,
mutually_exclusive
=
None
,
required_together
=
None
,
required_one_of
=
None
,
add_file_common_args
=
False
,

supports_check_mode
=
False
,
required_if
=
None
,
required_by
=
None
)
¶
Common code for quickly building an ansible module in Python
(although you can write modules with anything that can return JSON).
See
Ansible module development: getting started
for a general introduction
and
Ansible module architecture
for more detailed explanation.
add_path_info
(
kwargs
)
¶
for results that are files, supplement the info about the file
in the return path with stats about the file path.
atomic_move
(
src
,
dest
,
unsafe_writes
=
False
)
¶
atomically move src to dest, copying attributes from dest, returns true on success
it uses os.rename to ensure this as it is an atomic operation, rest of the function is
to work around limitations, corner cases and ensure selinux context is saved if possible
backup_local
(
fn
)
¶
make a date-marked backup of the specified file, return True or False on success or failure
boolean
(
arg
)
¶
Convert the argument to a boolean

digest_from_file
(
filename
,
algorithm
)
¶
Return hex digest of local file for a digest_method specified by name, or None if file is not present.
exit_json
(
**
kwargs
)
¶
return from the module, without error
fail_json
(
**
kwargs
)
¶
return from the module, with an error message
get_bin_path
(
arg
,
required
=
False
,
opt_dirs
=
None
)
¶
Find system executable in PATH.
Parameters
arg
? The executable to find.
required
? if executable is not found and required is
True
, fail_json
opt_dirs
? optional list of directories to search in addition to
PATH
Returns
if found return full path; otherwise return None
is_executable
(
path
)
¶
is the given path executable?

Parameters
path
? The path of the file to check.
Limitations:
Does not account for FSACLs.
Most times we really want to know ?Can the current user execute this
file?. This function does not tell us that, only if any execute bit is set.
is_special_selinux_path
(
path
)
¶
Returns a tuple containing (True, selinux_context) if the given path is on a
NFS or other ?special? fs mount point, otherwise the return will be (False, None).
load_file_common_arguments
(
params
)
¶
many modules deal with files, this encapsulates common
options that the file module accepts such that it is directly
available to all modules and they can share code.
md5
(
filename
)
¶
Return MD5 hex digest of local file using digest_from_file().
Do not use this function unless you have no other choice for:
Optional backwards compatibility
Compatibility with a third party protocol
This function will not work on systems complying with FIPS-140-2.
Most uses of this function can use the module.sha1 function instead.
preserved_copy
(
src
,
dest
)
¶
Copy a file with preserved ownership, permissions and context
run_command
(
args
,
check_rc
=
False
,
close_fds
=
True
,
executable

=
None
,
data
=
None
,
binary_data
=
False
,
path_prefix
=
None
,
cwd
=
None
,
use_unsafe_shell
=
False
,
prompt_regex
=
None
,
environ_update
=
None
,
umask
=
None
,
encoding
=
'utf-8'
,
errors
=
'surrogate_or_strict'
,
expand_user_and_vars
=
True
,
pass_fds
=
None
,
before_communicate_callback
=
None

)
¶
Execute a command, returns rc, stdout, and stderr.
Parameters
args
? is the command to run
* If args is a list, the command will be run with shell=False.
* If args is a string and use_unsafe_shell=False it will split args to a list and run with shell=False
* If args is a string and use_unsafe_shell=True it runs with shell=True.
Kw check_rc
Whether to call fail_json in case of non zero RC.
Default False
Kw close_fds
See documentation for subprocess.Popen(). Default True
Kw executable
See documentation for subprocess.Popen(). Default None
Kw data
If given, information to write to the stdin of the command
Kw binary_data
If False, append a newline to the data. Default False
Kw path_prefix
If given, additional path to find the command in.
This adds to the PATH environment variable so helper commands in
the same directory can also be found
Kw cwd
If given, working directory to run the command inside
Kw use_unsafe_shell
See
args
parameter. Default False
Kw prompt_regex
Regex string (not a compiled regex) which can be
used to detect prompts in the stdout which would otherwise cause
the execution to hang (especially if no input data is specified)
Kw environ_update
dictionary to
update
os.environ with
Kw umask
Umask to be used when running the command. Default None
Kw encoding
Since we return native strings, on python3 we need to
know the encoding to use to transform from bytes to text. If you
want to always get bytes back, use encoding=None. The default is
?utf-8?. This does not affect transformation of strings given as
args.
Kw errors
Since we return native strings, on python3 we need to
transform stdout and stderr from bytes to text. If the bytes are
undecodable in the
encoding
specified, then use this error
handler to deal with them. The default is
surrogate_or_strict

which means that the bytes will be decoded using the
surrogateescape error handler if available (available on all
python3 versions we support) otherwise a UnicodeError traceback
will be raised. This does not affect transformations of strings
given as args.
Kw expand_user_and_vars
When
use_unsafe_shell=False
this argument
dictates whether
~
is expanded in paths and environment variables
are expanded before running the command. When
True
a string such as
$SHELL
will be expanded regardless of escaping. When
False
and
use_unsafe_shell=False
no path or variable expansion will be done.
Kw pass_fds
When running on Python 3 this argument
dictates which file descriptors should be passed
to an underlying
Popen
constructor. On Python 2, this will
set
close_fds
to False.
Kw before_communicate_callback
This function will be called
after
Popen
object will be created
but before communicating to the process.
(
Popen
object will be passed to callback as a first argument)
Returns
A 3-tuple of return code (integer), stdout (native string),
and stderr (native string). On python2, stdout and stderr are both
byte strings. On python3, stdout and stderr are text strings converted
according to the encoding and errors parameters. If you want byte
strings on python3, use encoding=None to turn decoding to text off.
sha1
(
filename
)
¶
Return SHA1 hex digest of local file using digest_from_file().
sha256
(
filename

)
¶
Return SHA-256 hex digest of local file using digest_from_file().
Basic
¶
To use this functionality, include
import
ansible.module_utils.basic
in your module.
exception
ansible.module_utils.basic.
AnsibleFallbackNotFound
¶
ansible.module_utils.basic.
env_fallback
(
*
args
,
**
kwargs
)
¶
Load value from environment
ansible.module_utils.basic.
get_all_subclasses
(
cls
)
¶
Deprecated
: Use ansible.module_utils.common._utils.get_all_subclasses instead
ansible.module_utils.basic.
get_platform
(
)
¶
Deprecated
Use
platform.system()
directly.
Returns
Name of the platform the module is running on in a native string
Returns a native string that labels the platform (?Linux?, ?Solaris?, etc). Currently, this is
the result of calling
platform.system()
.
ansible.module_utils.basic.
heuristic_log_sanitize
(
data
,
no_log_values
=

None
)
¶
Remove strings that look like passwords from log messages
ansible.module_utils.basic.
load_platform_subclass
(
cls
,
*
args
,
**
kwargs
)
¶
Deprecated
: Use ansible.module_utils.common.sys_info.get_platform_subclass instead
ansible.module_utils.basic.
remove_values
(
value
,
no_log_strings
)
¶
Remove strings in no_log_strings from value. If value is a container
type, then remove a lot more.
Use of deferred_removals exists, rather than a pure recursive solution,
because of the potential to hit the maximum recursion depth when dealing with
large amounts of data (see issue #24560).
ansible.module_utils.basic.
sanitize_keys
(
obj
,
no_log_strings
,
ignore_keys
=
frozenset({})
)
¶
Sanitize the keys in a container object by removing no_log values from key names.
This is a companion function to the
remove_values()
function. Similar to that function,
we make use of deferred_removals to avoid hitting maximum recursion depth in cases of
large data structures.
Parameters
obj
? The container object to sanitize. Non-container objects are returned unmodified.
no_log_strings
? A set of string values we do not want logged.

ignore_keys
? A set of string values of keys to not sanitize.
Returns
An object with sanitized keys.
Next
Previous

Last updated on May 27, 2022.

Content from: https://docs.ansible.com/ansible/devel/community/communication.html

Communicating with the Ansible community ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Community Code of Conduct
Developer Certificate Of Origin
Communicating with the Ansible community
Code of Conduct
Forum
Real-time chat
Working groups
Ansible Community Topics
Ansible Automation Platform support questions
How can I help?
Other ways to get involved
Contributor path
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics

Communicating with the Ansible community
?
Code of Conduct
Forum
The Bullhorn
Regional and Language-specific channels
Real-time chat
General channels
Working group-specific channels
Meetings on Matrix
Working groups
Requesting a forum group
Requesting a community collection repository
Ansible Community Topics
Ansible Automation Platform support questions
Code of Conduct
?
All interactions within the Ansible Community are governed by our
Community Code of Conduct
. Please read and understand it before participating.
Forum
?

The
Ansible Forum
is the
default and recommended
starting point for most community interactions. It?s ideal for:
Asking questions and seeking help.
Participating in development discussions.
Learning about events and news.
To get started:
Register:
Sign up
to create an account and join the community.
Explore topics:
Browse by
categories
and
tags
to discover discussions, or simply start a new topic of your own.
Stay updated:
Subscribe to specific categories or tags that interest you. Just click the
bell
icon in the top-right corner of the relevant category or tag page and select your notification preference.
Explore
forum groups
that match your interests. Joining a group often automatically subscribes you to related posts.
The Bullhorn
?
The Bullhorn
is our community?s weekly newsletter, published directly in the Forum:
Subscribe:
Click the
bell
button under the
Bullhorn category
description, then select
Watching
.
Submit News:
See the
About the Newsletter category post
for submission guidelines.
Questions about the newsletter:
Ask us in the
Ansible Social room on Matrix
.
Regional and Language-specific channels
?
Communicate in your preferred language by visiting the
International Communities forum category
. Current subcategories include:
Deutsche (German)
Español (Spanish)
Français (French)
Italiano (Italian)

Norsk (Norwegian)
Português (Portuguese)
Join an
Ansible Meetup
near you.
For details on requesting a new language subcategory, see the
About the International Communities category post
.
Real-time chat
?
For real-time interactions, the Ansible community uses the
Matrix protocol
.
Note
The
Forum
is our default communication platform. We recommend engaging there before considering other options like Matrix.
To join the community on Matrix:
Get a Matrix account:
From
Matrix.org
or any other Matrix homeserver.
Choose a Matrix client:
We recommend
Element Webchat
.
Join rooms:
Use the links in the
General channels
or
Working groups
to join specific rooms.
For more information, see the community-hosted
Matrix FAQ
.
You can add Matrix shields to your repository?s
README.md
using the shield in the
community-topics
repository as a template.
Note
IRC channels are no longer official communication channels. Use the Forum and Matrix instead.
General channels
?
The clickable links below take you directly to the Matrix room in your browser. Room/channel information is also given for use in other clients:
Community social room & posting news for the Bullhorn newsletter
General usage & support questions
Developer & code-related topics
Community & collections related topics
Working group-specific channels
?
Many working groups have dedicated chat channels. See the
Working groups

for details.
Meetings on Matrix
?
The Ansible community holds regular meetings on Matrix. All interested individuals are invited to participate.
Check the
meeting schedule and agenda page
for more information.
Working groups
?
Working Groups enable Ansible community members to self-organize around specific interests.
Find a complete list of groups and their communication channels within the
Forum groups
.
Requesting a forum group
?
To request a new working group:
First, check if there is no appropriate
Forum group
you can join instead of starting a new one.
Review the
things you can ask for post
regarding working groups.
Submit your request in the
forum topic
.
If a Matrix chat channel is also needed, consult the
Ansible Community Matrix FAQ
.
Requesting a community collection repository
?
Working groups are often built around Ansible community collections. You can use a repository under your organization or request one under
ansible-collections
on the forum. Create a topic in the
Project Discussions category and the ?coll-repo-request? tag
.
Ansible Community Topics
?
The
Ansible Community Steering Committee
uses the
Forum
for asynchronous discussions and voting on community topics.
For more information, see:
Creating new policy proposals & inclusion requests
Community topics workflow
Community topics on the Forum
Ansible Automation Platform support questions
?
Red Hat Ansible
Automation Platform
is a subscription service providing support, certified content, and tooling for Ansible, including content management, a controller, UI and REST API.
For questions related to Ansible Automation Platform, visit

Red Hat support
instead of community communication platforms.
Previous
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/devel/community/contributions_collections.html

Ansible Collections Contributor Guide ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Contributing to the Ansible Documentation
Other Tools and Programs
Working with the Ansible collection repositories
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide

Overview
Feedback and communications
Keeping informed
Communication and Working Groups
Collection infrastructure
Python Compatibility
Standards for developing module and plugin utilities
Repository structure requirements
Contributor Workflow
Naming
Collection licensing requirements
Contributor License Agreements
Repository management
CI Testing
When moving modules between collections
Development conventions
Collection Dependencies
Other requirements
Guidelines for collection maintainers
Maintainer responsibilities
Expanding the collection community
Maintaining good collection documentation
Ansible Collection Maintenance and Workflow
Stepping down as a collection maintainer
Releasing collections
Contributing to Ansible-maintained Collections
Ansible-maintained collections
Deciding where your contribution belongs
Requirements to merge your PR
Ansible Community Steering Committee
Steering Committee mission and responsibilities
Steering Committee membership guidelines
Steering Committee past members
Community topics workflow
Contributing to the Ansible Documentation
Editing docs directly on GitHub
Reviewing or solving open issues
Reviewing open PRs
Opening a new issue and/or PR
Verifying your documentation PR
Joining the documentation working group
Other Tools and Programs
Popular editors
Tools for validating playbooks
Collection development tools
Other tools
If you have a specific Ansible interest or expertise (for example, VMware, Linode, and so on), consider joining a working group
.
Working with the Ansible collection repositories
?
How can I find
editors, linters, and other tools
that will support my Ansible development efforts?

Where can I find guidance on
coding in Ansible
?
How do I
create a collection
?
How do I
rebase my PR
?
How do I learn about Ansible?s
testing (CI) process
?
How do I
deprecate a module
?
See
Collection developer tutorials
for a quick introduction on how to develop and test your collection contributions.
Previous
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/devel/community/contributor_path.html

Contributor path ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Contributor path
Determine your area of interest
Find the corresponding project
Learn
Specific knowledge for code developers
Making your first contribution
Continue to contribute
Teach others
Become a collection maintainer
Become a steering committee member
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy

Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Community Guide
Contributor path
Edit on GitHub
Contributor path
?
This section describes the contributor?s journey from the beginning to becoming a leader who helps shape the future of Ansible. You can use this path as a roadmap for your long-term participation.
Any contribution to the project, even a small one, is very welcome and valuable. Any contribution counts, whether it is feedback on an issue, a pull request, a topic or documentation change, or a coding contribution. When you contribute regularly, your proficiency and judgment in the related area increase and, along with this, the importance of your presence in the project.
Determine your area of interest
Find the corresponding project
Learn
Specific knowledge for code developers
Making your first contribution
Continue to contribute
Teach others
Become a collection maintainer
Become a steering committee member
Determine your area of interest
?
First, determine areas that are interesting to you. Consider your current experience and what you?d like to gain. For example, if you use a specific collection, have a look there. See
How can I help?
for more ideas on how to help.
Find the corresponding project
?

These are multiple community projects in the Ansible ecosystem you could contribute to:
Ansible Core
Collections
AWX
Galaxy
ansible-lint
Molecule
Learn
?
The required skillset depends on the area of interest and the project you?ll be working on. Remember that the best way to learn is by doing.
Specific knowledge for code developers
?
Code development requires the most technical knowledge. Let?s sort out what an Ansible developer should learn.
You should understand at least the
basics
of the following tools:
Python programming language
Git
GitHub collaborative development model through forks and pull requests
You can learn these tools more in-depth when working on your first contributions.
Each Ansible project has its own set of contributor guidelines. Familiarize yourself with these as you prepare your first contributions.
Ansible Core development
.
Ansible collection development
and the collection-level contributor guidelines in the collection repository.
Making your first contribution
?
You can find some ideas on how you can contribute in
How can I help?
.
If you are interested in contributing to collections, take a look at
collection contributions
and the
collection repository
?s
README
and
CONTRIBUTING
files. To make your first experience as smooth as possible, read the repository documentation carefully, then ask the repository maintainers for guidance if you have any questions.
Take a look at GitHub issues labeled with the
easyfix
and
good_first_issue
labels for:
Ansible collections repositories
All other Ansible projects
Issues labeled with the
docs
label in
Ansible collections
and

other
Ansible projects can be also good to start with.
When you choose an issue to work on, add a comment directly on the GitHub issue to say you are looking at it and let others know to avoid conflicting work.
You can also ask for help in a comment if you need it.
Continue to contribute
?
We don?t expect everybody to know everything. Start small, think big. When you contribute regularly, your proficiency and judgment in the related area will improve quickly and, along with this, the importance of your presence in the project.
See
Communicating with the Ansible community
for ways to communicate and engage with the Ansible community, including working group meetings, accessing the Bullhorn news bulletin, and upcoming contributor summits.
Teach others
?
Share your experience with other contributors through
improving documentation
, answering questions from other contributors and users on
Matrix/Libera.Chat IRC
, giving advice on issues and pull requests, and discussing topics on the
Forum
.
Become a collection maintainer
?
If you are a code contributor to a collection, you can get extended permissions in the repository and become a maintainer. A collection maintainer is a contributor trusted by the community who makes significant and regular contributions to the project and showed themselves as a specialist in the related area. See
Guidelines for collection maintainers
for details.
For some collections that use the
collection bot
, such as
community.general
and
community.network
, you can have different levels of access and permissions:
File-level permissions: the stage prior to becoming a collection maintainer. The file is usually a module or plugin. File maintainers have indirect commit rights.
Supershipit permissions: similar to being a file maintainer but the scope where a maintainer has the indirect commit is the whole repository.
Triage access to the repository: allows contributors to manage issues and pull requests.
Write access to the repository also known as
commit
: allows contributors to merge pull requests to the development branch as well as perform all the other activities listed in the
Guidelines for collection maintainers
.
For information about permission levels, see the
GitHub official documentation
.
Become a steering committee member
?
Note
You do NOT have to be a programmer to become a steering committee member.

The
Steering Committee
member status reflects the highest level of trust and allows contributors to lead the project by making important decisions for the Ansible project. The Committee members are community leaders who shape the project?s future and the future of automation in the IT world in general.
To reach the status, as the current Committee members did before getting it, along with the things mentioned in this document, you should:
Subscribe to, comment on, and vote on the
community topics<creating_community_topic>
.
Propose your topics.
If time permits, join the
Community meetings
. Note this is
NOT
a requirement.
Previous
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/devel/community/steering/community_steering_co

Steering Committee mission and responsibilities ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Steering Committee mission and responsibilities
Steering Committee responsibilities
Current Steering Committee members
Creating new policy proposals & inclusion requests
Community Working Group meetings
Steering Committee membership guidelines
Steering Committee past members
Community topics workflow
Contributing to the Ansible Documentation
Other Tools and Programs
Working with the Ansible collection repositories
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide

Steering Committee mission and responsibilities

?

The Steering Committee mission is to provide continuity, guidance, and suggestions to the Ansible community to ensure the delivery and high quality of the Ansible package. In addition, the committee helps decide the technical direction of the Ansible project. It is responsible for approving new proposals and policies in the community, package, and community collections world, new community collection-inclusion requests, and other technical aspects regarding inclusion and packaging.

The Committee should reflect the scope and breadth of the Ansible community.

Steering Committee responsibilities

?

The Committee:

Designs policies and procedures for the community collections world.

Votes on approval changes to established policies and procedures.

Reviews community collections for compliance with the policies.

Helps create and define roadmaps for our deliverables such as the

ansible
package, major community collections, and documentation.
Reviews community collections submitted for inclusion in the Ansible package and decides whether to include them or not.
Review other proposals of importance that need the Committee?s attention and provide feedback.
Current Steering Committee members
?
The following table lists the current Steering Committee members. See
Steering Committee past members
for a list of past members.
The Steering Committee itself is also a group in the forum (
@SteeringCommittee
).
Current Steering committee members
?
Name
GitHub
Forum
Start year
Alexei Znamensky
russoz
russoz
2022
Alicia Cozine
acozine
acozine
2021
Andrew Klychkov
Andersson007
Andersson007
2021
Brian Scholer
briantist
briantist
2022
Core Team repr.
See Forum
2024
Felix Fontein
felixfontein
felixfontein
2021
James Cassell
jamescassell
cassel
2021
John Barker
gundalow
gundalow
2021
Mario Lenz
mariolenz
mariolenz
2022

Markus Bergholz
markuman
markuman
2022
Maxwell G
gotmax23
gotmax23
2022
Sorin Sbarnea
ssbarnea
ssbarnea
2021
John Barker (
gundalow
) has been elected by the Committee as its
Chairperson
.
Committee members are selected based on their active contribution to the Ansible Project and its community. See
Steering Committee membership guidelines
to learn details.
Creating new policy proposals & inclusion requests
?
The Committee uses the
Ansible Forum
to asynchronously discuss with the Community and vote on the proposals in corresponding
community topics
.
You can
create a community topic
(make sure you use the
Project
Discussions
category and the
community-wg
tag) if you want to discuss an idea that impacts any of the following:
Ansible Community
Community collection best practices and requirements
Community collection inclusion/exclusion policy and workflow
The Community governance
Other proposals of importance that need the Committee?s or overall Ansible community attention
To request changes to the inclusion policy and
Ansible community package collections requirements
:
Submit a new pull request against the document by clicking the
Edit
button on its web page.
Create a
community topic
containing the rationale for the proposed changes.
To submit new collections for inclusion into the Ansible package:
Submit the new collection inclusion requests through a new discussion in the
ansible-inclusion
repository.
Depending on a topic you want to discuss with the Community and the Committee, as you prepare your proposal,

please consider the requirements established by:
Community Code of Conduct
.
Ansible community package collections requirements
.
Ansible Collection Inclusion Checklist
.
Ansible Community Package Collections Removal Process
.
Community topics workflow
?
The Committee uses the
Community topics workflow
to asynchronously discuss and vote on the
community topics
.
The quorum, the minimum number of Committee members who must vote on a topic in order for a decision to be officially made, is half of the whole number of the Committee members. If the quorum number contains a fractional part, it is rounded up to the next whole number. For example, if there are thirteen members currently in the committee, the quorum will be seven.
Votes must always have ?no change? as an option.
In case of equal numbers of votes for and against a topic, the chairperson?s vote will break the tie. For example, if there are six votes for and six votes against a topic, and the chairperson?s vote is among those six which are for the topic, the final decision will be positive. If the chairperson has not voted yet, other members ask them to vote.
For votes with more than two options, one choice must have at least half of the votes. If two choices happen to both have half of the votes, the chairperson?s vote will break the tie. If no choice has at least half of the votes, the vote choices have to be adjusted so that a majority can be found for a choice in a new vote.
Community topics triage
?
The Committee conducts a triage of
community topics
periodically (every three to six months).
The triage goals are:
Sparking interest for forgotten topics.
Identifying and closing irrelevant topics, for example, when the reason of the topic does not exist anymore or the topic is out of the Committee responsibilities scope.
Identifying and closing topics that the Community are not interested in discussing. As indicators, it can be absence of comments or no activity in comments, at least, for the last six months.
Identifying and closing topics that were solved and implemented but not closed (in this case, such a topic can be closed on the spot with a comment that it has been implemented).
Identifying topics that have been in pending state for a long time, for example, when it is waiting for actions from someone for several months or when the topics were solved but not implemented.
A person starting the triage:
Identifies the topics mentioned above.
Creates a special triage topic containing an enumerated list of the topics-candidates for closing.
Establishes a vote date considering a number of topics, their complexity and comment-history size giving the Community sufficient time to go through and discuss them.
The Community and the Committee vote on each topic-candidate listed in the triage topic whether to close it or keep it open.
Collection inclusion requests workflow
?
When reviewing community collection
inclusion requests
, the Committee members check if a collection adheres to the

Ansible community package collections requirements
.
Note
The Steering Committee can reject a collection inclusion request or exclude a collection from the Ansible package even when the collection satisfies the requirements if the Steering Committee agrees that presence of the collection will significantly deteriorate the Ansible package user experience or the package build process. In case of rejection/removal, the collection maintainers receive comprehensive feedback from the Committee explaining the reasons of starting the process. In case the reasons are fixable, the feedback will contain information what the maintainers need to change.
A Committee member who conducts the inclusion review copies the
Ansible community collection checklist
into a corresponding
discussion
.
In the course of the review, the Committee member marks items as completed or leaves a comment saying whether the reviewer expects an issue to be addressed or whether it is optional (for example, it could be
MUST FIX:
<what> or
SHOULD FIX:
<what> under an item).
For a collection to be included in the Ansible community package, the collection:
MUST be reviewed and approved as compliant with the requirements by at least two Steering Committee members.
At least one of the reviews checks compliance with the entire checklist.
All subsequent reviews can focus only on compliance with documentation and development conventions.
Reviewers must not be involved significantly in development of the collection. They MUST declare any potential conflict of interest (for example, being friends/relatives/coworkers of the maintainers/authors, being users of the collection, or having contributed to that collection recently or in the past).
After the collection gets two Committee member approvals, a Committee member creates a
community topic
linked to the corresponding inclusion request. The issue?s description says that the collection has been approved by the Committee and establishes a date (a week by default) when the inclusion decision will be considered made.
The inclusion automatically gets suspended if the Committee members raise concerns or start another inclusion review within this time period.
When there are no more objections or ongoing inclusion reviews, the inclusion date gets prolonged for another week.
If the inclusion has not been suspended by the established date, the inclusion request is considered successfully resolved. In this case, a Committee member:
Declares the decision in the topic and in the inclusion request.
Moves the request to the
Resolved
reviews
category.
Adds the collection to the
ansible.in
file in a corresponding directory of the
ansible-build-data repository
.
Announces the inclusion through the
Bullhorn newsletter
.
Closes the topic.
Collection exclusion workflow
?
The Committee uses the
Ansible Community Package Collections Removal Process
to remove collections not satisfying the

Ansible community package collections requirements
from the Ansible package.
Community Working Group meetings
?
See the Community Working Group meeting
schedule
. Meeting summaries are posted on the
Forum
.
Note
Participation in the Community Working Group meetings is optional for Committee members. Decisions on community topics
are made asynchronously in the topics themselves.
Previous
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/devel/dev_guide/developing_collections_distributin

Distributing collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing collections
Distributing collections
Edit on GitHub
Distributing collections
?
A collection is a distribution format for Ansible content. A typical collection contains modules and other plugins that address a set of related use cases. For example, a collection might automate administering a particular database. A collection can also contain roles and playbooks.
Note
Before distributing your collection, ensure you have updated the
galaxy.yml
file.
See
Collection structure
for details.
To distribute your collection and allow others to use it, you can publish your collection on one or more
distribution server
. Distribution servers include:
Distribution server
Collections accepted
Ansible Galaxy
All collections
Pulp 3 Galaxy
All collections, supports signed collections
Red Hat Automation Hub
Only collections certified by Red Hat, supports signed collections
Privately hosted Automation Hub
Collections authorized by the owners
Distributing collections involves four major steps:
Initial configuration of your distribution server or servers
Building your collection tarball
Preparing to publish your collection
Publishing your collection
Initial configuration of your distribution server or servers
Creating a namespace
Getting your API token
Specifying your API token and distribution server
Building your collection tarball

Ignoring files and folders
Signing a collection
Preparing to publish your collection
Installing your collection locally
Reviewing your collection
Understanding collection versioning
Publishing your collection
Publishing a collection from the command line
Publishing a collection from the website
Initial configuration of your distribution server or servers
?
Configure a connection to one or more distribution servers so you can publish collections there. You only need to configure each distribution server once. You must repeat the other steps (building your collection tarball, preparing to publish, and publishing your collection) every time you publish a new collection or a new version of an existing collection.
Create a namespace on each distribution server you want to use.
Get an API token for each distribution server you want to use.
Specify the API token for each distribution server you want to use.
Creating a namespace
?
You must upload your collection into a namespace on each distribution server. If you have a login for Ansible Galaxy, your Ansible Galaxy username is usually also an Ansible Galaxy namespace.
Warning
Namespaces on Ansible Galaxy cannot include hyphens. If you have a login for Ansible Galaxy that includes a hyphen, your Galaxy username is not also a Galaxy namespace. For example,
awesome-user
is a valid username for Ansible Galaxy, but it is not a valid namespace.
You can create additional namespaces on Ansible Galaxy if you choose. For Red Hat Automation Hub and private Automation Hub you must create a namespace before you can upload your collection. To create a namespace:
To create a namespace on Galaxy, see
Galaxy namespaces
on the Galaxy docsite for details.
To create a namespace on Red Hat Automation Hub, see the
Red Hat automation content documentation
.
Specify the namespace in the
galaxy.yml
file for each collection. For more information on the
galaxy.yml
file, see
Collection Galaxy metadata structure
.
Getting your API token
?
An API token authenticates your connection to each distribution server. You need a separate API token for each distribution server. Use the correct API token to connect to each distribution server securely and protect your content.
To get your API token:
To get an API token for Galaxy, see
the Galaxy documentation
.
To get an API token for Automation Hub, see the
Red Hat automation content documentation
.
Specifying your API token and distribution server

?
Each time you publish a collection, you must specify the API token and the distribution server to create a secure connection. You have two options for specifying the token and distribution server:
You can configure the token in configuration, as part of a
galaxy_server_list
entry in your
ansible.cfg
file. Using configuration is the most secure option.
You can pass the token at the command line as an argument to the
ansible-galaxy
command. If you pass the token at the command line, you can specify the server at the command line, by using the default setting, or by setting the server in configuration. Passing the token at the command line is insecure, because typing secrets at the command line may expose them to other users on the system.
Specifying the token and distribution server in configuration
?
By default, Ansible Galaxy is configured as the only distribution server. You can add other distribution servers and specify your API token or tokens in configuration by editing the
galaxy_server_list
section of your
ansible.cfg
file. This is the most secure way to manage authentication for distribution servers. Specify a URL and token for each server. For example:
[galaxy]
server_list
=
release_galaxy
[galaxy_server.release_galaxy]
url
=
https://galaxy.ansible.com/
token
=
abcdefghijklmnopqrtuvwxyz
You cannot use
apt-key
with any servers defined in your
galaxy_server_list
. See
Configuring the ansible-galaxy client
for complete details.
Specifying the token at the command line
?
You can specify the API token at the command line using the
--token
argument of the
ansible-galaxy
command. There are three ways to specify the distribution server when passing the token at the command line:
using the
--server
argument of the
ansible-galaxy
command
relying on the default (
https://galaxy.ansible.com

)
setting a server in configuration by creating a
GALAXY_SERVER
setting in your
ansible.cfg
file
For example:
ansible-galaxy
collection
publish
path/to/my_namespace-my_collection-1.0.0.tar.gz
--token
abcdefghijklmnopqrtuvwxyz
Warning
Using the
--token
argument is insecure. Passing secrets at the command line may expose them to others on the system.
Building your collection tarball
?
After configuring one or more distribution servers, build a collection tarball. The collection tarball is the published artifact, the object that you upload and other users download to install your collection. To build a collection tarball:
Review all settings in your
galaxy.yml
file. See
Collection Galaxy metadata structure
for details. Ensure you have updated the version number. Each time you publish your collection, it must have a new version number. You cannot make changes to existing versions of your collection on a distribution server. If you try to upload the same collection version more than once, the distribution server returns the error
Code:
conflict.collection_exists
. Collections follow semantic versioning rules. For more information on versions, see
Understanding collection versioning
. For more information on the
galaxy.yml
file, see
Collection Galaxy metadata structure
.
Run
ansible-galaxy
collection
build
from inside the top-level directory of the collection. For example:
collection_dir#>
ansible-galaxy
collection
build
This command builds a tarball of the collection in the current directory, which you can upload to your selected distribution server:
my_collection/
???
galaxy.yml
???
...
???

my_namespace-my_collection-1.0.0.tar.gz
???
...
Note
To reduce the size of collections, certain files and folders are excluded from the collection tarball by default. See
Ignoring files and folders
if your collection directory contains other files you want to exclude.
The current Galaxy maximum tarball size is 2 MB.
You can upload your tarball to one or more distribution servers. You can also distribute your collection locally by copying the tarball to install your collection directly on target systems.
Ignoring files and folders
?
You can exclude files from your collection with either
build_ignore
or
Manifest Directives
. For more information on the
galaxy.yml
file, see
Collection Galaxy metadata structure
.
Include all, with explicit ignores
?
By default, the build step includes all the files in the collection directory in the tarball except for the following:
galaxy.yml
*.pyc
*.retry
tests/output
previously built tarballs in the root directory
various version control directories such as
.git/
To exclude other files and folders from your collection tarball, set a list of file glob-like patterns in the
build_ignore
key in the collection?s
galaxy.yml
file. These patterns use the following special characters for wildcard matching:
*
: Matches everything
?
: Matches any single character
[seq]
: Matches any character in sequence
[!seq]
:Matches any character not in sequence
For example, to exclude the
sensitive
folder within the
playbooks
folder as well any
.tar.gz
archives, set the following in your
galaxy.yml
file:
build_ignore

:
-
playbooks/sensitive
-
'*.tar.gz'
Note
The
build_ignore
feature is only supported with
ansible-galaxy
collection
build
in Ansible 2.10 or newer.
Manifest Directives
?
New in version 2.14.
The
galaxy.yml
file supports manifest directives that are historically used in Python packaging, as described in
MANIFEST.in commands
.
Note
The use of
manifest
requires installing the optional
distlib
Python dependency.
Note
The
manifest
feature is only supported with
ansible-galaxy
collection
build
in
ansible-core
2.14 or newer, and is mutually exclusive with
build_ignore
.
For example, to exclude the
sensitive
folder within the
playbooks
folder as well as any
.tar.gz
archives, set the following in your
galaxy.yml
file:
manifest
:
directives
:
-
recursive-exclude playbooks/sensitive **

-
global-exclude *.tar.gz
By default, the
MANIFEST.in
style directives would exclude all files by default, but there are default directives in place. Those default directives are described below. To see the directives in use during build, pass
-vvv
with the
ansible-galaxy
collection
build
command.
include meta/*.yml
include *.txt *.md *.rst COPYING LICENSE
recursive-include tests **
recursive-include docs **.rst **.yml **.yaml **.json **.j2 **.txt
recursive-include roles **.yml **.yaml **.json **.j2
recursive-include playbooks **.yml **.yaml **.json
recursive-include changelogs **.yml **.yaml
recursive-include plugins */**.py
recursive-include plugins/become **.yml **.yaml
recursive-include plugins/cache **.yml **.yaml
recursive-include plugins/callback **.yml **.yaml
recursive-include plugins/cliconf **.yml **.yaml
recursive-include plugins/connection **.yml **.yaml
recursive-include plugins/filter **.yml **.yaml
recursive-include plugins/httpapi **.yml **.yaml
recursive-include plugins/inventory **.yml **.yaml
recursive-include plugins/lookup **.yml **.yaml
recursive-include plugins/netconf **.yml **.yaml
recursive-include plugins/shell **.yml **.yaml
recursive-include plugins/strategy **.yml **.yaml
recursive-include plugins/test **.yml **.yaml
recursive-include plugins/vars **.yml **.yaml
recursive-include plugins/modules **.ps1 **.yml **.yaml
recursive-include plugins/module_utils **.ps1 **.psm1 **.cs
manifest.directives from galaxy.yml inserted here
exclude galaxy.yml galaxy.yaml MANIFEST.json FILES.json <namespace>-<name>-*.tar.gz
recursive-exclude tests/output **
global-exclude /.* /__pycache__
Note
<namespace>-<name>-*.tar.gz
is expanded with the actual
namespace
and
name
.
The
manifest.directives
supplied in
galaxy.yml
are inserted after the default includes and before the default excludes.
To enable the use of manifest directives without supplying your own, insert either
manifest:

{}
or
manifest:
null
in the
galaxy.yml
file and remove any use of
build_ignore
.
If the default manifest directives do not meet your needs, you can set
manifest.omit_default_directives
to a value of
true
in
galaxy.yml
. You then must specify a full compliment of manifest directives in
galaxy.yml
. The defaults documented above are a good starting point.
Below is an example where the default directives are not included.
manifest
:
directives
:
-
include meta/runtime.yml
-
include README.md LICENSE
-
recursive-include plugins */**.py
-
exclude galaxy.yml MANIFEST.json FILES.json <namespace>-<name>-*.tar.gz
-
recursive-exclude tests/output **
omit_default_directives
:
true
Signing a collection
?
You can include a GnuPG signature with your collection on a
Pulp 3 Galaxy
server. See
Enabling collection signing
for details.
You can manually generate detached signatures for a collection using the
gpg
CLI using the following step. This step assume you have generated a GPG private key, but do not cover this process.
ansible-galaxy
collection
build
tar
-Oxzf
namespace-name-1.0.0.tar.gz
MANIFEST.json
|

gpg
--output
namespace-name-1.0.0.asc
--detach-sign
--armor
--local-user

-
Preparing to publish your collection
?
Each time you publish your collection, you must create a
new version
on the distribution server. After you publish a version of a collection, you cannot delete or modify that version. To avoid unnecessary extra versions, check your collection for bugs, typos, and other issues locally before publishing:
Install the collection locally.
Review the locally installed collection before publishing a new version.
Installing your collection locally
?
You have two options for installing your collection locally:
Install your collection locally from the tarball.
Install your collection locally from your Git repository.
Installing your collection locally from the tarball
?
To install your collection locally from the tarball, run
ansible-galaxy
collection
install
and specify the collection tarball. You can optionally specify a location using the
-p
flag. For example:
collection_dir#>
ansible-galaxy
collection
install
my_namespace-my_collection-1.0.0.tar.gz
-p
./collections
Install the tarball into a directory configured in
COLLECTIONS_PATHS
so Ansible can easily find and load the collection. If you do not specify a path value,
ansible-galaxy
collection
install
installs the collection in the first path defined in
COLLECTIONS_PATHS
.
Installing your collection locally from a Git repository
?
To install your collection locally from a Git repository, specify the repository and the branch you want to install:
collection_dir#>
ansible-galaxy
collection
install
git+https://github.com/org/repo.git,devel

You can install a collection from a git repository instead of from Galaxy or Automation Hub. As a developer, installing from a git repository lets you review your collection before you create the tarball and publish the collection. As a user, installing from a git repository lets you use collections or versions that are not in Galaxy or Automation Hub yet. This functionality is meant as a minimal shortcut for developers of content as previously described, and git repositories may not support the full set of features from the
ansible-galaxy
CLI. In complex cases, a more flexible option may be to
git
clone
the repository into the correct file structure of the collection installation directory.
The repository must contain a
galaxy.yml
or
MANIFEST.json
file. This file provides metadata such as the version number and namespace of the collection.
Installing a collection from a git repository at the command line
?
To install a collection from a git repository at the command line, use the URI of the repository instead of a collection name or path to a
tar.gz
file. Use the prefix
git+
, unless you?re using SSH authentication with the user
git
(for example,
git@github.com:ansible-collections/ansible.windows.git
). You can specify a branch, commit, or tag using the comma-separated
git commit-ish
syntax.
For example:
Install a collection in a repository using the latest commit on the branch 'devel'
ansible-galaxy
collection
install
git+https://github.com/organization/repo_name.git,devel
Install a collection from a private GitHub repository
ansible-galaxy
collection
install

:organization/repo_name.git
Install a collection from a local git repository
ansible-galaxy
collection
install
git+file:///home/user/path/to/repo_name.git
Warning
Embedding credentials into a git URI is not secure. Use safe authentication options to prevent your credentials from being exposed in logs or elsewhere.
Use
SSH
authentication
Use
netrc

authentication
Use
http.extraHeader
in your git configuration
Use
url.<base>.pushInsteadOf
in your git configuration
Specifying the collection location within the git repository
?
When you install a collection from a git repository, Ansible uses the collection
galaxy.yml
or
MANIFEST.json
metadata file to build the collection. By default, Ansible searches two paths for collection
galaxy.yml
or
MANIFEST.json
metadata files:
The top level of the repository.
Each directory in the repository path (one level deep).
If a
galaxy.yml
or
MANIFEST.json
file exists in the top level of the repository, Ansible uses the collection metadata in that file to install an individual collection.
??? galaxy.yml
??? plugins/
?   ??? lookup/
?   ??? modules/
?   ??? module_utils/
???? README.md
If a
galaxy.yml
or
MANIFEST.json
file exists in one or more directories in the repository path (one level deep), Ansible installs each directory with a metadata file as a collection. For example, Ansible installs both collection1 and collection2 from this repository structure by default:
??? collection1
?   ??? docs/
?   ??? galaxy.yml
?   ??? plugins/
?   ??? inventory/
?   ??? modules/
??? collection2
    ??? docs/
    ??? galaxy.yml
    ??? plugins/
    | ??? filter/
    | ??? modules/
    ??? roles/
If you have a different repository structure or only want to install a subset of collections, you can add a fragment to the end of your URI (before the optional comma-separated version) to indicate the location of the metadata file or files. The

path should be a directory, not the metadata file itself. For example, to install only collection2 from the example repository with two collections:
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/collection2/
In some repositories, the main directory corresponds to the namespace:
namespace/
??? collectionA/
| ??? docs/
| ??? galaxy.yml
| ??? plugins/
| ? ??? README.md
| ? ??? modules/
| ??? README.md
| ??? roles/
??? collectionB/
??? docs/
??? galaxy.yml
??? plugins/
? ??? connection/
? ??? modules/
??? README.md
??? roles/
You can install all collections in this repository, or install one collection from a specific commit:
Install all collections in the namespace
ansible-galaxy
collection
install
git+https://github.com/organization/repo_name.git#/namespace/
Install an individual collection using a specific commit
ansible-galaxy
collection
install
git+https://github.com/organization/repo_name.git#/namespace/collectionA/,7b60ddc245bc416b72d8ea6ed7b799885110f5e5
Reviewing your collection
?
Review the collection:
Run a playbook that uses the modules and plugins in your collection. Verify that new features and functionality work as expected. For examples and more details see
Using collections
.
Check the documentation for typos.
Check that the version number of your tarball is higher than the latest published version on the distribution server or servers.
If you find any issues, fix them and rebuild the collection tarball.
Understanding collection versioning
?
The only way to change a collection is to release a new version. The latest version of a collection (by highest version number) is the version displayed everywhere in Galaxy and Automation Hub. Users can still download older versions.
Follow semantic versioning when setting the version for your collection. In summary:
Increment the major version number,
x
of
x.y.z
, for an incompatible API change.

Increment the minor version number,
y
of
x.y.z
, for new functionality in a backwards compatible manner (for example new modules/plugins, parameters, return values).
Increment the patch version number,
z
of
x.y.z
, for backwards compatible bug fixes.
Read the official
Semantic Versioning
documentation for details and examples.
Publishing your collection
?
The last step in distributing your collection is publishing the tarball to Ansible Galaxy, Red Hat Automation Hub, or a privately hosted Automation Hub instance. You can publish your collection in two ways:
from the command line using the
ansible-galaxy
collection
publish
command
from the website of the distribution server (Galaxy, Automation Hub) itself
Publishing a collection from the command line
?
To upload the collection tarball from the command line using
ansible-galaxy
:
ansible-galaxy
collection
publish
path/to/my_namespace-my_collection-1.0.0.tar.gz
Note
This ansible-galaxy command assumes you have retrieved and stored your API token in configuration. See
Specifying your API token and distribution server
for details.
The
ansible-galaxy
collection
publish
command triggers an import process, just as if you uploaded the collection through the Galaxy website. The command waits until the import process completes before reporting the status back. If you want to continue without waiting for the import result, use the
--no-wait
argument and manually look at the import progress in your
My Imports
page.
Publishing a collection from the website
?
See the
Galaxy documentation
to learn how to publish your collection directly on the Galaxy website.
See also
Using Ansible collections

Learn how to install and use collections.
Collection Galaxy metadata structure
Table of fields used in the
galaxy.yml
file
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/devel/getting_started_ee/index.html

Getting started with Execution Environments ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Introduction to Execution Environments
Setting up your environment
Building your first Execution Environment
Running your EE
Running Ansible with the community EE image
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords

Getting started with Execution Environments
?
You can run Ansible automation in containers, like any other modern software application.
Ansible uses container images known as Execution Environments (EE) that act as control nodes.
EEs remove complexity to scale out automation projects and make things like deployment operations much more straightforward.
An Execution Environment image contains the following packages as standard:
ansible-core
ansible-runner
Python
Ansible content dependencies
In addition to the standard packages, an EE can also contain:
one or more Ansible collections and their dependencies
other custom components
This getting started guide shows you how to build and test a simple Execution Environment.
The resulting container image represents an Ansible control node that contains:
standard EE packages
community.postgresql
collection
psycopg2-binary
Python package

Last updated on Oct 19, 2025.

Content from: https://docs.ansible.com/ansible/index.html

Ansible Documentation ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Documentation
Ansible Documentation
?
Welcome to Ansible community documentation!
This documentation covers the version of Ansible noted in the upper left corner of this page.
We maintain multiple versions of Ansible and of the documentation, so please be sure you are using the version of the documentation that covers the version of Ansible you?re using.
For recent features, we note the version of Ansible where the feature was added.
Ansible releases a new major release approximately twice a year.
The core application evolves somewhat conservatively, valuing simplicity in language design and setup.
Contributors develop and change modules and plugins, hosted in collections, much more quickly.
Ansible getting started
Getting started with Ansible
Introduction to Ansible
Start automating with Ansible
Building an inventory
Creating a playbook
Ansible concepts
Getting started with Execution Environments
Introduction to Execution Environments
Setting up your environment
Building your first Execution Environment
Running your EE
Running Ansible with the community EE image
Installation, Upgrade & Configuration
Installation Guide
Installing Ansible
Installing Ansible on specific operating systems
Configuring Ansible
Ansible Porting Guides
Ansible 12 Porting Guide
Ansible 11 Porting Guide
Ansible 10 Porting Guide
Ansible 9 Porting Guide
Ansible 8 Porting Guide
Ansible 7 Porting Guide
Ansible 6 Porting Guide
Ansible 5 Porting Guide
Ansible 4 Porting Guide

YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
Ansible project 12.0
Ansible project 11.0
Ansible project 10.0
Ansible project 9.0
Ansible project 8.0
Ansible project 7.0
Ansible project 6.0
Ansible project 5.0
Ansible project 4.0
Ansible project 3.0
Ansible project 2.10
Older Roadmaps
ansible-core Roadmaps
Ansible-core 2.19
Ansible-core 2.18
Ansible-core 2.17
Ansible-core 2.16
Ansible-core 2.15
Ansible-core 2.14
Ansible-core 2.13
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/collections/all_plugins.html

Indexes of all modules and plugins ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Index of all Become Plugins
Index of all Cache Plugins
Index of all Callback Plugins
Index of all Cliconf Plugins
Index of all Connection Plugins
Index of all Filter Plugins

Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/collections/community/general/merge_variab

community.general.merge_variables lookup ? Merge variables whose names match a given pattern ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Collections in the Amazon Namespace
Collections in the Ansible Namespace
Collections in the Arista Namespace
Collections in the Awx Namespace
Collections in the Azure Namespace
Collections in the Check_point Namespace

community.general.merge_variables lookup ? Merge variables whose names match a given pattern

?
Note
This lookup plugin is part of the
community.general collection
(version 11.4.0).
You might already have this collection installed if you are using the
ansible
package.
It is not included in
ansible-core
.
To check whether it is installed, run
ansible-galaxy
collection
list
.
To install it, use:
ansible-galaxy
collection
install
community.general
.
To use it in a playbook, specify:
community.general.merge_variables
.
New in community.general 6.5.0
Synopsis
Terms
Keyword parameters
Notes
Examples
Return Value
Synopsis
?
This lookup returns the merged result of all variables in scope that match the given prefixes, suffixes, or regular expressions, optionally.
Terms
?
Parameter
Comments
Terms
list
/
elements=string
/
required
Depending on the value of
pattern_type
, this is a list of prefixes, suffixes, or regular expressions that is used to match all variables that should be merged.
Keyword parameters
?
This describes keyword parameters of the lookup. These are the values
key1=value1
,

key2=value2
and so on in the following
examples:
lookup('community.general.merge_variables',
key1=value1,
key2=value2,
...)
and
query('community.general.merge_variables',
key1=value1,
key2=value2,
...)
Parameter
Comments
groups
list
/
elements=string
added in community.general 8.5.0
Search for variables across hosts that belong to the given groups. This allows to collect configuration pieces across different hosts (for example a service on a host with its database on another host).
initial_value
any
An initial value to start with.
override
string
Return an error, print a warning or ignore it when a key is overwritten.
The default behavior
error
makes the plugin fail when a key would be overwritten.
When
warn
and
ignore
are used, note that it is important to know that the variables are sorted by name before being merged. Keys for later variables in this order overwrite keys of the same name for variables earlier in this order. To avoid potential confusion, better use
override=error
whenever possible.
Choices:
"error"
? (default)
"warn"
"ignore"
Configuration:
INI entry:
[merge_variables_lookup]
override
=
error
Environment variable:
ANSIBLE_MERGE_VARIABLES_OVERRIDE
pattern_type
string

Change the way of searching for the specified pattern.
Choices:
"prefix"
"suffix"
"regex"
? (default)
Configuration:
INI entry:
[merge_variables_lookup]
pattern_type
=
regex
Environment variable:
ANSIBLE_MERGE_VARIABLES_PATTERN_TYPE
Note
Configuration entries listed above for each entry type (Ansible variable, environment variable, and so on) have a low to high priority order.
For example, a variable that is lower in the list will override a variable that is higher up.
The entry types are also ordered by precedence from low to high priority order.
For example, an ansible.cfg entry (further up in the list) is overwritten by an Ansible variable (further down in the list).
Notes
?
Note
When keyword and positional parameters are used together, positional parameters must be listed before keyword parameters:
lookup('community.general.merge_variables',
term1,
term2,
key1=value1,
key2=value2)
and
query('community.general.merge_variables',
term1,
term2,
key1=value1,
key2=value2)
Examples
?
Some example variables, they can be defined anywhere as long as they are in scope
test_init_list
:
-
"list
init
item
1"
-
"list
init
item
2"
testa__test_list
:
-

"test
a
item
1"
testb__test_list
:
-
"test
b
item
1"
testa__test_dict
:
ports
:
-
1
testb__test_dict
:
ports
:
-
3
Merge variables that end with '__test_dict' and store the result in a variable 'example_a'
example_a
:
"
{{
lookup
(
'community.general.merge_variables'
,
'__test_dict'
,
pattern_type
=
'suffix'
)
}}
"
The variable example_a now contains:
ports:
- 1
- 3
Merge variables that match the '^.+__test_list$' regular expression, starting with an initial value and store the
result in a variable 'example_b'
example_b
:
"
{{
lookup
(
'community.general.merge_variables'
,

'^.+__test_list$'
,
initial_value
=
test_init_list
)
}}
"
The variable example_b now contains:
- "list init item 1"
- "list init item 2"
- "test a item 1"
- "test b item 1"
Return Value
?
Key
Description
Return value
any
In case the search matches list items, a list is returned. In case the search matches dicts, a dict is returned.
Returned:
success
Authors
?
Roy Lenferink (@rlenferink)
Mark Ettema (@m-a-r-k-e)
Alexander Petrenz (@alpex8)
Collection links
?
Issue Tracker
Repository (Sources)
Ask for help
Submit a bug report
Request a feature
Communication
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/collections/index.html

Collection Index ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Collections in the Amazon Namespace
Collections in the Ansible Namespace
Collections in the Arista Namespace
Collections in the Awx Namespace
Collections in the Azure Namespace
Collections in the Check_point Namespace
Collections in the Chocolatey Namespace

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Collection Index
Collection Index
?
These are the collections with docs hosted on
docs.ansible.com
.
amazon.aws
ansible.builtin
ansible.netcommon
ansible.posix
ansible.utils
ansible.windows
arista.eos
awx.awx
azure.azcollection
check_point.mgmt
chocolatey.chocolatey
cisco.aci
cisco.dnac
cisco.intersight
cisco.ios
cisco.iosxr
cisco.meraki
cisco.mso
cisco.nxos
cisco.ucs
cloudscale_ch.cloud
community.aws
community.ciscosmb
community.crypto
community.digitalocean
community.dns
community.docker
community.general
community.grafana
community.hashi_vault
community.hrobot
community.library_inventory_filtering_v1
community.libvirt

community.mongodb
community.mysql
community.okd
community.postgresql
community.proxmox
community.proxysql
community.rabbitmq
community.routeros
community.sap_libs
community.sops
community.vmware
community.windows
community.zabbix
containers.podman
cyberark.conjur
cyberark.pas
dellemc.enterprise_sonic
dellemc.openmanage
dellemc.powerflex
dellemc.unity
f5networks.f5_modules
fortinet.fortimanager
fortinet.fortios
google.cloud
grafana.grafana
hetzner.hcloud
hitachivantara.vspone_block
hitachivantara.vspone_object
ibm.qradar
DEPRECATED: REMOVED IN 13
ibm.storage_virtualize
ieisystem.inmanage
infinidat.infinibox
infoblox.nios_modules
inspur.ispim
junipernetworks.junos
kaytus.ksmanage
kubernetes.core
kubevirt.core
lowlydba.sqlserver
microsoft.ad
microsoft.iis
netapp.cloudmanager
netapp.ontap
netapp.storagegrid
netapp_eseries.santricity
netbox.netbox
ngine_io.cloudstack
openstack.cloud
ovirt.ovirt
purestorage.flasharray
purestorage.flashblade
ravendb.ravendb
splunk.es

telekom_mms.icinga_director
theforeman.foreman
vmware.vmware
vmware.vmware_rest
vultr.cloud
vyos.vyos
wti.remote
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/collections_guide/index.html

Using Ansible collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Installing collections
Removing a collection
Downloading collections
Listing collections
Verifying collections
Using collections in a playbook
Collections index
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index

Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Using Ansible collections
Edit on GitHub
Using Ansible collections
?
Note
Making Open Source More Inclusive
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see
our CTO Chris Wright?s message
.
Welcome to the Ansible guide for working with collections.
Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.
You can install and use collections through a distribution server, such as Ansible Galaxy, or a Pulp 3 Galaxy server.
Installing collections
Installing collections in containers
Installing collections with
ansible-galaxy
Installing collections with signature verification
Installing an older version of a collection
Install multiple collections with a requirements file
Downloading a collection for offline use
Installing collections adjacent to playbooks
Installing a collection from source files
Installing a collection from a Git repository
Configuring the
ansible-galaxy
client
Removing a collection
Downloading collections
Listing collections
Verifying collections

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/code_of_conduct.html

Community Code of Conduct ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Community Code of Conduct
Anti-harassment policy
Policy violations
Developer Certificate Of Origin
Communicating with the Ansible community
How can I help?
Other ways to get involved
Contributor path
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide

Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Community Guide
Getting started
Community Code of Conduct
Edit on GitHub
Community Code of Conduct
?
Topics
Community Code of Conduct
Anti-harassment policy
Policy violations
Every community can be strengthened by a diverse variety of viewpoints, insights, opinions, skillsets, and skill levels. However, with diversity comes the potential for disagreement and miscommunication. The purpose of this Code of Conduct is to ensure that disagreements and differences of opinion are conducted respectfully and on their own merits, without personal attacks or other behavior that might create an unsafe or unwelcoming environment.
These policies are not designed to be a comprehensive set of Things You Cannot Do. We ask that you treat your fellow community members with respect and courtesy, and in general, Don?t Be A Jerk. This Code of Conduct is meant to be followed in spirit as much as in letter and is not exhaustive.
All Ansible events and participants therein are governed by this Code of Conduct and anti-harassment policy. We expect organizers to enforce these guidelines throughout all events, and we expect attendees, speakers, sponsors, and volunteers to help ensure a safe environment for our whole community. Specifically, this Code of Conduct covers participation in all Ansible-related forums and mailing lists, code and documentation contributions, public chat (Matrix, IRC), private correspondence, and public meetings.
Ansible community members are?
Considerate
Contributions of every kind have far-ranging consequences. Just as your work depends on the work of others, decisions you make surrounding your contributions to the Ansible

community will affect your fellow community members. You are strongly encouraged to take those consequences into account while making decisions.

Patient

Asynchronous communication can come with its own frustrations, even in the most responsive of communities. Please remember that our community is largely built on volunteered time, and that questions, contributions, and requests for support may take some time to receive a response. Repeated ?bumps? or ?reminders? in rapid succession are not good displays of patience. Additionally, it is considered poor manners to ping a specific person with general questions. Pose your question to the community as a whole, and wait patiently for a response.

Respectful

Every community inevitably has disagreements, but remember that it is possible to disagree respectfully and courteously. Disagreements are never an excuse for rudeness, hostility, threatening behavior, abuse (verbal or physical), or personal attacks.

Kind

Everyone should feel welcome in the Ansible community, regardless of their background. Please be courteous, respectful and polite to fellow community members. Do not make or post offensive comments related to skill level, gender, gender identity or expression, sexual orientation, disability, physical appearance, body size, race, or religion. Sexualized images or imagery, real or implied violence, intimidation, oppression, stalking, sustained disruption of activities, publishing the personal information of others without explicit permission to do so, unwanted physical contact, and unwelcome sexual attention are all strictly prohibited. Additionally, you are encouraged not to make assumptions about the background or identity of your fellow community members.

Inquisitive

The only stupid question is the one that does not get asked. We encourage our users to ask early and ask often. Rather than asking whether you can ask a question (the answer is always yes!), instead, simply ask your question. You are encouraged to provide as many specifics as possible. Code snippets in the form of Gists or other paste site links are almost always needed in order to get the most helpful answers. Refrain from pasting multiple lines of code directly into the chat channels - instead use gist.github.com or another paste site to provide code snippets.

Helpful

The Ansible community is committed to being a welcoming environment for all users, regardless of skill level. We were all beginners once upon a time, and our community cannot grow without an environment where new users feel safe and comfortable asking questions. It can become frustrating to answer the same questions repeatedly; however, community members are expected to remain courteous and helpful to all users equally, regardless of skill or knowledge level. Avoid providing responses that prioritize snideness and snark over useful information. At the same time, everyone is expected to read the provided documentation thoroughly. We are happy to answer questions, provide strategic guidance, and suggest effective workflows, but we are not here to do your job for you.

Anti-harassment policy

?

Harassment includes (but is not limited to) all of the following behaviors:

Offensive comments related to gender (including gender expression and identity), age, sexual orientation, disability, physical appearance, body size, race, and religion

Derogatory terminology including words commonly known to be slurs

Posting sexualized images or imagery in public spaces

Deliberate intimidation

Stalking

Posting others? personal information without explicit permission

Sustained disruption of talks or other events

Inappropriate physical contact

Unwelcome sexual attention

Participants asked to stop any harassing behavior are expected to comply immediately.

Sponsors are also subject to the anti-harassment policy. In particular, sponsors should not use sexualized images, activities, or other material. Meetup organizing staff and other volunteer organizers should not use sexualized attire or otherwise create a sexualized environment at community events.

In addition to the behaviors outlined above, continuing to behave a certain way after you have been asked to stop also constitutes harassment, even if that behavior is not specifically outlined in this policy. It is considerate and respectful to stop doing something after you have been asked to stop, and all community members are expected to comply with such requests immediately.

Policy violations

?

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting

codeofconduct

@

ansible

.

com

, to anyone with administrative power in community chat (Admins or Moderators on Matrix, ops on IRC), or to the local organizers of an event. Meetup organizers are encouraged to prominently display points of contact for reporting unacceptable behavior at local events.

If a participant engages in harassing behavior, the meetup organizers may take any action they deem appropriate. These actions may include but are not limited to warning the offender, expelling the offender from the event, and barring the offender from future community events.

Organizers will be happy to help participants contact security or local law enforcement, provide escorts to an alternate location, or otherwise assist those experiencing harassment to feel safe for the duration of the meetup. We value the safety and well-being of our community members and want everyone to feel welcome at our events, both online and offline.

We expect all participants, organizers, speakers, and attendees to follow these policies at all of our event venues and event-related social events.

 Our Code of Conduct was adapted from Codes of Conduct of other open source projects, including:

Contributor Covenant

Elastic

The Fedora Project

OpenStack

Puppet Labs

Ubuntu

Previous

Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/collection_contributors/collection

Releasing collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Maintainer responsibilities
Expanding the collection community
Maintaining good collection documentation
Ansible Collection Maintenance and Workflow
Stepping down as a collection maintainer
Releasing collections
Preparing to release a collection
Collection versioning and deprecation
Collection changelogs
Options for releasing a collection
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Contributing to the Ansible Documentation
Other Tools and Programs
Working with the Ansible collection repositories
ansible-core Contributors Guide

Releasing collections

?

Collection maintainers release all supported stable versions of the collections regularly, provided that there have been enough changes merged to release.

?

The collections under the
ansible-collections organization
follow
semantic versioning
when releasing. See
Collection versioning and deprecation
for details.
To prepare for a release, a collection must have:
A publicly available policy of releasing, versioning, and deprecation. This can be, for example, written in its README or in a dedicated pinned issue.
A pinned issue when its release managers inform the community about planned or completed releases. This can be combined with the release policy issue mentioned above.
A
changelog
.
Releases of the collection tagged in the collection?s repository.
CI pipelines up and running. This can be implemented by using GitHub Actions, Azure Pipelines, Zuul.
All CI tests running against a commit that releases the collection. If they do not pass, the collection MUST NOT be released.
See
Including a collection in Ansible
if you plan on adding a new collection to the Ansible package.
Note
Your collection must pass
ansible-test
sanity
tests. See
Testing collections
for details.
Collection versioning and deprecation
?
Note
Collections MUST adhere to
semantic versioning
.
To preserve backward compatibility for users, every Ansible minor version series (5.1.x, 5.2.x, and so on) will keep the major version of a collection constant. For example, if Ansible 5.0.0 includes
community.general
4.0.2, then each Ansible 5.X.x release will include the latest
community.general
4.y.z release available at build time. Ansible 5.x.x will
never
include a
community.general
5.y.x release, even if it is available. Major collection version changes will be included in the next Ansible major release (6.0.0 in this case).
Ensure that the current major release of your collection included in 6.0.0 receives at least bugfixes as long as new Ansible 6.X.X releases are produced.
Since new minor releases are included, you can include new features, modules and plugins. You must make sure that you do not break backward compatibility. See
semantic versioning
. for more details. This means in particular:
You can fix bugs in
patch

releases but not add new features or deprecate things.
You can add new features and deprecate things in
minor
releases, but not remove things or change the behavior of existing features.
You can only remove things or make breaking changes in
major
releases.
Ensure that if a deprecation is added in a collection version that is included in 5.x.y, the removal itself will only happen in a collection version included in 7.0.0 or later.
Ensure that the policy of releasing, versioning, and deprecation is announced to contributors and users in some way. For an example of how to do this, see
the announcement in community.general
. You could also do this in the collection README file.
Collection changelogs
?
Collections MUST include a changelog. To give a consistent feel for changelogs across collections and ensure changelogs exist for collections included in the
ansible
package, we suggest you use
antsibull-changelog
to maintain and generate this.
Before releasing, verify the following for your changelogs:
All merged pull requests since the last release, except ones related to documentation and new modules/plugins, have changelog fragments
.
New module and plugin pull requests, except jinja2 test and filter plugins, do
not
need a changelog fragment, they are auto-detected by the changelog generator by their
version_added
value.
All the fragments follow the
changelog entry format
.
Options for releasing a collection
?
There are several approaches on how to release a collection. If you are not aware of which approach to use, ask in the
#ansible-community
IRC channel or the
community
Matrix channel.
This section assumes that publishing the collection is done with
Zuul
and that
antsibull-changelog
is used for the changelog.
Releasing without release branches
?
Use releasing without release branches when:
There are no prior major releases of the collection.
There are no breaking changes introduced since the
1.0.0
release of the collection.
See
Releasing collections without release branches

for details.
When there is a need to introduce breaking changes, you can switch to the next approach.
Hybrid approach
?
In this approach, releases for the current major version are made from the
main
branch, while new releases for older major versions are made from release branches for these versions.
Releasing with release branches
?
Use releasing with release branches when breaking changes have been introduced. This approach is usually only used by the large community collections,
community.general
and
community.network
.
See
Releasing collections with release branches
for details.
Releasing collections without release branches
Releasing collections with release branches
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/communication.html

Communicating with the Ansible community ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Community Code of Conduct
Developer Certificate Of Origin
Communicating with the Ansible community
Code of Conduct
Forum
Real-time chat
Working groups
Ansible Community Topics
Ansible Automation Platform support questions
How can I help?
Other ways to get involved
Contributor path
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics

Communicating with the Ansible community
?
Code of Conduct
Forum
The Bullhorn
Regional and Language-specific channels
Real-time chat
General channels
Working group-specific channels
Meetings on Matrix
Working groups
Requesting a forum group
Requesting a community collection repository
Ansible Community Topics
Ansible Automation Platform support questions
Code of Conduct
?
All interactions within the Ansible Community are governed by our
Community Code of Conduct
. Please read and understand it before participating.
Forum
?

The
Ansible Forum
is the
default and recommended
starting point for most community interactions. It?s ideal for:
Asking questions and seeking help.
Participating in development discussions.
Learning about events and news.
To get started:
Register:
Sign up
to create an account and join the community.
Explore topics:
Browse by
categories
and
tags
to discover discussions, or simply start a new topic of your own.
Stay updated:
Subscribe to specific categories or tags that interest you. Just click the
bell
icon in the top-right corner of the relevant category or tag page and select your notification preference.
Explore
forum groups
that match your interests. Joining a group often automatically subscribes you to related posts.
The Bullhorn
?
The Bullhorn
is our community?s weekly newsletter, published directly in the Forum:
Subscribe:
Click the
bell
button under the
Bullhorn category
description, then select
Watching
.
Submit News:
See the
About the Newsletter category post
for submission guidelines.
Questions about the newsletter:
Ask us in the
Ansible Social room on Matrix
.
Regional and Language-specific channels
?
Communicate in your preferred language by visiting the
International Communities forum category
. Current subcategories include:
Deutsche (German)
Español (Spanish)
Français (French)
Italiano (Italian)

Norsk (Norwegian)
Português (Portuguese)
Join an
Ansible Meetup
near you.
For details on requesting a new language subcategory, see the
About the International Communities category post
.
Real-time chat
?
For real-time interactions, the Ansible community uses the
Matrix protocol
.
Note
The
Forum
is our default communication platform. We recommend engaging there before considering other options like Matrix.
To join the community on Matrix:
Get a Matrix account:
From
Matrix.org
or any other Matrix homeserver.
Choose a Matrix client:
We recommend
Element Webchat
.
Join rooms:
Use the links in the
General channels
or
Working groups
to join specific rooms.
For more information, see the community-hosted
Matrix FAQ
.
You can add Matrix shields to your repository?s
README.md
using the shield in the
community-topics
repository as a template.
Note
IRC channels are no longer official communication channels. Use the Forum and Matrix instead.
General channels
?
The clickable links below take you directly to the Matrix room in your browser. Room/channel information is also given for use in other clients:
Community social room & posting news for the Bullhorn newsletter
General usage & support questions
Developer & code-related topics
Community & collections related topics
Working group-specific channels
?
Many working groups have dedicated chat channels. See the
Working groups

for details.
Meetings on Matrix
?
The Ansible community holds regular meetings on Matrix. All interested individuals are invited to participate.
Check the
meeting schedule and agenda page
for more information.
Working groups
?
Working Groups enable Ansible community members to self-organize around specific interests.
Find a complete list of groups and their communication channels within the
Forum groups
.
Requesting a forum group
?
To request a new working group:
First, check if there is no appropriate
Forum group
you can join instead of starting a new one.
Review the
things you can ask for post
regarding working groups.
Submit your request in the
forum topic
.
If a Matrix chat channel is also needed, consult the
Ansible Community Matrix FAQ
.
Requesting a community collection repository
?
Working groups are often built around Ansible community collections. You can use a repository under your organization or request one under
ansible-collections
on the forum. Create a topic in the
Project Discussions category and the ?coll-repo-request? tag
.
Ansible Community Topics
?
The
Ansible Community Steering Committee
uses the
Forum
for asynchronous discussions and voting on community topics.
For more information, see:
Creating new policy proposals & inclusion requests
Community topics workflow
Community topics on the Forum
Ansible Automation Platform support questions
?
Red Hat Ansible
Automation Platform
is a subscription service providing support, certified content, and tooling for Ansible, including content management, a controller, UI and REST API.
For questions related to Ansible Automation Platform, visit

Red Hat support
instead of community communication platforms.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/contributions_collections.html

Ansible Collections Contributor Guide ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Contributing to the Ansible Documentation
Other Tools and Programs
Working with the Ansible collection repositories
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide

Overview
Feedback and communications
Keeping informed
Communication and Working Groups
Collection infrastructure
Python Compatibility
Standards for developing module and plugin utilities
Repository structure requirements
Contributor Workflow
Naming
Collection licensing requirements
Contributor License Agreements
Repository management
CI Testing
When moving modules between collections
Development conventions
Collection Dependencies
Other requirements
Guidelines for collection maintainers
Maintainer responsibilities
Expanding the collection community
Maintaining good collection documentation
Ansible Collection Maintenance and Workflow
Stepping down as a collection maintainer
Releasing collections
Contributing to Ansible-maintained Collections
Ansible-maintained collections
Deciding where your contribution belongs
Requirements to merge your PR
Ansible Community Steering Committee
Steering Committee mission and responsibilities
Steering Committee membership guidelines
Steering Committee past members
Community topics workflow
Contributing to the Ansible Documentation
Editing docs directly on GitHub
Reviewing or solving open issues
Reviewing open PRs
Opening a new issue and/or PR
Verifying your documentation PR
Joining the documentation working group
Other Tools and Programs
Popular editors
Tools for validating playbooks
Collection development tools
Other tools
If you have a specific Ansible interest or expertise (for example, VMware, Linode, and so on), consider joining a working group
.
Working with the Ansible collection repositories
?
How can I find
editors, linters, and other tools
that will support my Ansible development efforts?

Where can I find guidance on
coding in Ansible
?
How do I
create a collection
?
How do I
rebase my PR
?
How do I learn about Ansible?s
testing (CI) process
?
How do I
deprecate a module
?
See
Collection developer tutorials
for a quick introduction on how to develop and test your collection contributions.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/contributor_path.html

Contributor path ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Contributor path
Determine your area of interest
Find the corresponding project
Learn
Specific knowledge for code developers
Making your first contribution
Continue to contribute
Teach others
Become a collection maintainer
Become a steering committee member
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy

Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Community Guide
Contributor path
Edit on GitHub
Contributor path
?
This section describes the contributor?s journey from the beginning to becoming a leader who helps shape the future of Ansible. You can use this path as a roadmap for your long-term participation.
Any contribution to the project, even a small one, is very welcome and valuable. Any contribution counts, whether it is feedback on an issue, a pull request, a topic or documentation change, or a coding contribution. When you contribute regularly, your proficiency and judgment in the related area increase and, along with this, the importance of your presence in the project.
Determine your area of interest
Find the corresponding project
Learn
Specific knowledge for code developers
Making your first contribution
Continue to contribute
Teach others
Become a collection maintainer
Become a steering committee member
Determine your area of interest
?
First, determine areas that are interesting to you. Consider your current experience and what you?d like to gain. For example, if you use a specific collection, have a look there. See
How can I help?
for more ideas on how to help.
Find the corresponding project
?

These are multiple community projects in the Ansible ecosystem you could contribute to:
Ansible Core
Collections
AWX
Galaxy
ansible-lint
Molecule
Learn
?
The required skillset depends on the area of interest and the project you?ll be working on. Remember that the best way to learn is by doing.
Specific knowledge for code developers
?
Code development requires the most technical knowledge. Let?s sort out what an Ansible developer should learn.
You should understand at least the
basics
of the following tools:
Python programming language
Git
GitHub collaborative development model through forks and pull requests
You can learn these tools more in-depth when working on your first contributions.
Each Ansible project has its own set of contributor guidelines. Familiarize yourself with these as you prepare your first contributions.
Ansible Core development
.
Ansible collection development
and the collection-level contributor guidelines in the collection repository.
Making your first contribution
?
You can find some ideas on how you can contribute in
How can I help?
.
If you are interested in contributing to collections, take a look at
collection contributions
and the
collection repository
?s
README
and
CONTRIBUTING
files. To make your first experience as smooth as possible, read the repository documentation carefully, then ask the repository maintainers for guidance if you have any questions.
Take a look at GitHub issues labeled with the
easyfix
and
good_first_issue
labels for:
Ansible collections repositories
All other Ansible projects
Issues labeled with the
docs
label in
Ansible collections
and

other
Ansible projects can be also good to start with.
When you choose an issue to work on, add a comment directly on the GitHub issue to say you are looking at it and let others know to avoid conflicting work.
You can also ask for help in a comment if you need it.
Continue to contribute
?
We don?t expect everybody to know everything. Start small, think big. When you contribute regularly, your proficiency and judgment in the related area will improve quickly and, along with this, the importance of your presence in the project.
See
Communicating with the Ansible community
for ways to communicate and engage with the Ansible community, including working group meetings, accessing the Bullhorn news bulletin, and upcoming contributor summits.
Teach others
?
Share your experience with other contributors through
improving documentation
, answering questions from other contributors and users on
Matrix/Libera.Chat IRC
, giving advice on issues and pull requests, and discussing topics on the
Forum
.
Become a collection maintainer
?
If you are a code contributor to a collection, you can get extended permissions in the repository and become a maintainer. A collection maintainer is a contributor trusted by the community who makes significant and regular contributions to the project and showed themselves as a specialist in the related area. See
Guidelines for collection maintainers
for details.
For some collections that use the
collection bot
, such as
community.general
and
community.network
, you can have different levels of access and permissions:
File-level permissions: the stage prior to becoming a collection maintainer. The file is usually a module or plugin. File maintainers have indirect commit rights.
Supershipit permissions: similar to being a file maintainer but the scope where a maintainer has the indirect commit is the whole repository.
Triage access to the repository: allows contributors to manage issues and pull requests.
Write access to the repository also known as
commit
: allows contributors to merge pull requests to the development branch as well as perform all the other activities listed in the
Guidelines for collection maintainers
.
For information about permission levels, see the
GitHub official documentation
.
Become a steering committee member
?
Note
You do NOT have to be a programmer to become a steering committee member.

The
Steering Committee
member status reflects the highest level of trust and allows contributors to lead the project by making important decisions for the Ansible project. The Committee members are community leaders who shape the project?s future and the future of automation in the IT world in general.
To reach the status, as the current Committee members did before getting it, along with the things mentioned in this document, you should:
Subscribe to, comment on, and vote on the
community topics<creating_community_topic>
.
Propose your topics.
If time permits, join the
Community meetings
. Note this is
NOT
a requirement.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/how_can_I_help.html

How can I help? ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Getting started
Community Code of Conduct
Developer Certificate Of Origin
Communicating with the Ansible community
How can I help?
Become a power user
Ask and answer questions online
Review, fix, and maintain the documentation
Participate in your local meetup
File and verify issues
Review and submit pull requests
Become a collection maintainer
Join a working group
Teach Ansible to others
Social media
Other ways to get involved
Contributor path
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios

How can I help?
?
Become a power user
Ask and answer questions online
Review, fix, and maintain the documentation
Participate in your local meetup
File and verify issues
Review and submit pull requests
Become a collection maintainer
Join a working group
Teach Ansible to others
Social media
Thanks for being interested in helping the Ansible project!
There are many ways to help the Ansible project?but first, please read and understand the
Community Code of Conduct
.
Become a power user
?

A great way to help the Ansible project is to become a power user:
Use Ansible everywhere you can
Take tutorials and classes
Read the
official documentation
Study some of the
many excellent books
about Ansible
Get certified
.
When you become a power user, your ability and opportunities to help the Ansible project in other ways will multiply quickly.
Ask and answer questions online
?
There are many online platforms where Ansible users and contributors ask and answer questions including the
Forum
.
Reach out and communicate with your fellow Ansible enthusiasts.
You can find the official
Ansible communication channels
.
Review, fix, and maintain the documentation
?
Typos are everywhere, even in the Ansible documentation. We work hard to keep the documentation up-to-date, but you may also find outdated examples. We offer easy ways to
report and/or fix documentation errors
.
Participate in your local meetup
?
There are Ansible meetups
all over the world
. Join your local meetup. Attend regularly. Ask good questions. Volunteer to give a presentation about how you use Ansible.
If there is no meetup near you, we are happy to help you
start one
.
File and verify issues
?
All software has bugs, and Ansible is no exception. When you find a bug, you can help tremendously by telling us about it:
Filing
issues for ansible-core
.
Filing
issues for collections
.
If the bug you found already exists in an issue, you can help by verifying the behavior of the reported bug with a comment in that issue, or by reporting any additional information.
Review and submit pull requests
?
As you become more familiar with how Ansible works, you may be able to fix issues or develop new features yourself. If you think you have a fix for a bug in Ansible, or if you have a new feature that you would like to share with millions of Ansible users, read all about the
development process

to learn how to get your code accepted into Ansible.
You can also get started with solving GitHub issues labeled with the
easyfix
and
good_first_issue
labels for:
Ansible collections
All other Ansible projects
When you choose an issue to work on, add a comment directly on the GitHub issue to say you are looking at it and let others know to avoid conflicting work.
You can also ask for help in a comment if you need it.
For collections, refer to the
Creating your first collection pull request
page to learn how to quickly set up your local environment, test your changes, and submit a ready-for-review pull request.
Another good way to help is to review pull requests that other Ansible users have submitted. Ansible Core keeps a full list of
open pull requests by file
, so if a particular module or plugin interests you, you can easily keep track of all the relevant new pull requests and provide testing or feedback. Alternatively, you can review the pull requests for any collections that interest you. Click
Issue tracker
on the collection documentation page to find the issues and PRs for that collection.
Become a collection maintainer
?
Once you have learned about the development process and have contributed code to a collection, we encourage you to become a maintainer of that collection. There are hundreds of modules in dozens of Ansible collections, and the vast majority of them are written and maintained entirely by members of the Ansible community.
See
collection maintainer guidelines
to learn more about the responsibilities of being an Ansible collection maintainer.
Join a working group
?
Working groups are a way for Ansible community members to self-organize around particular topics of interest. We have working groups around various topics. To join or create a working group, please read the
Ansible Working Groups
.
Teach Ansible to others
?
We are working on a standardized
Ansible workshop
that can provide a good hands-on introduction to Ansible usage and concepts.
Social media
?
If you like Ansible and just want to spread the good word, feel free to share on your social media platform of choice, and let us know by using
@ansible
or
#ansible
. We?ll be looking for you.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/maintainers_guidelines.html

Maintainer responsibilities ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Maintainer responsibilities
Collection maintainer definition
Maintainer responsibilities
Becoming a maintainer
Communicating as a maintainer
Establishing working group communication
Participating in community topics
Expanding the collection community
Maintaining good collection documentation
Ansible Collection Maintenance and Workflow
Stepping down as a collection maintainer
Releasing collections
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Contributing to the Ansible Documentation
Other Tools and Programs

Working with the Ansible collection repositories
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Collections Contributor Guide
Guidelines for collection maintainers
Maintainer responsibilities
Edit on GitHub

Maintainer responsibilities

?

This document provides guidance for:

Contributors to collections who want to join maintainer teams.

Collection maintainers seeking to understand their roles better.

This document defines the role of an Ansible collection maintainer, outlines their responsibilities, and describes the process for becoming one.

Establishing working group communication
Participating in community topics
Collection maintainer definition
?
An Ansible collection maintainer, or simply maintainer, is a contributor who:
Makes significant and regular contributions to a project.
Demonstrates expertise in the area the collection automates.
Earns the community?s trust. To fulfill their duties, maintainers have
write
or higher access to the collection.
Maintainer responsibilities
?
Collection maintainers perform the following tasks:
Act in accordance with the
Community Code of Conduct
.
Subscribe to the repository they maintain. In GitHub, click
Watch > All activity
.
Keep the
README
, development guidelines, and other general
Maintaining good collection documentation
current.
Review and commit changes from other contributors using the
Review checklist for collection PRs
.
Backport
changes to stable branches.
Plan and perform releases
.
Ensure the collection adheres to the
Ansible community package collections requirements
.
Track changes announced through the
news-for-maintainers
forum tag. Click the
Bell
button to subscribe. Update the collection accordingly.
Build a healthy community
to increase the number of active contributors and maintainers for collections.
Multiple maintainers can divide these responsibilities among themselves.
Becoming a maintainer
?
If you are interested in becoming a maintainer and meet the
requirements
, nominate yourself. You can also nominate another person by following these steps:
Create a GitHub issue in the relevant repository.
If you receive no response, message the
Red Hat Ansible Community Engineering Team
on the
Ansible forum
.
Communicating as a maintainer

?
Maintainers communicate with the community through the channels listed in the
Ansible communication guide
.
Establishing working group communication
?
Working groups rely on efficient communication. As a maintainer, you can establish communication for your working groups using these techniques:
Find and join an existing
forum group
and use tags that suit your project.
If no suitable options exist,
request them
.
Provide working group details and chat room links in the contributor section of your project?s
README.md
.
Encourage contributors to join the forum group and use appropriate tags.
Participating in community topics
?
The Community and the
Steering Committee
discuss and vote on
community topics
asynchronously. These topics impact the entire project or its components, including collections and packaging.
Share your opinion and vote on the topics to help the community make informed decisions.
Expanding the collection community
?
You can expand the community around your collection in the following ways:
Explicitly state in your
README
that the collection welcomes new maintainers and contributors.
Give
newcomers a positive first experience
.
Invite contributors to join forum groups and subscribe to tags related to your project.
Maintain
good documentation
with guidelines for new contributors.
Make people feel welcome personally and individually. Greet and thank them.
Use labels to identify easy fixes and leave non-critical easy fixes to newcomers.
Offer help explicitly.
Include quick ways contributors can help and provide contributor documentation references in your
README
.
Be responsive in issues, pull requests (PRs), and other communication channels.
Conduct PR days regularly.
Maintain a zero-tolerance policy toward behavior that violates the
Community Code of Conduct
.
* Include information about how people can report code of conduct violations in your
README
and
CONTRIBUTING

files.
Look for new maintainers among active contributors.
Maintaining good collection documentation
?
Ensure the collection documentation meets these criteria:
It is up-to-date.
It matches the
Ansible documentation style guide
.
Collection module and plugin documentation adheres to the
Ansible documentation format
.
Collection user guides follow the
Collection documentation format
.
Repository files include at least a
README
and
CONTRIBUTING
file.
The
README
file contains all sections from
collection_template/README.md
.
The
CONTRIBUTING
file includes all details or links to details on how new or continuing contributors can contribute to your collection.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/community/maintainers_workflow.html

Ansible Collection Maintenance and Workflow ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
The Ansible Collections Development Cycle
Requesting changes to a collection
Creating your first collection pull request
Testing Collection Contributions
Review checklist for collection PRs
Ansible community package collections requirements
Guidelines for collection maintainers
Maintainer responsibilities
Expanding the collection community
Maintaining good collection documentation
Ansible Collection Maintenance and Workflow
Managing pull requests
Releasing a collection
Backporting
Including a collection in Ansible
Stepping down as a collection maintainer
Releasing collections
Contributing to Ansible-maintained Collections
Ansible Community Steering Committee
Contributing to the Ansible Documentation
Other Tools and Programs
Working with the Ansible collection repositories
ansible-core Contributors Guide

Edit on GitHub
Ansible Collection Maintenance and Workflow
?
Each collection community can set its own rules and workflows for managing pull requests (PRs), bug reports, documentation issues, feature requests, as well as for adding and replacing maintainers.
Collection maintainers have
write
or higher access to a collection, allowing them to merge pull requests and perform other administrative tasks.
Managing pull requests
?
Maintainers review and merge PRs according to the following guidelines:
Community Code of Conduct
Maintainer responsibilities
Committer guidelines

PR review checklist
Releasing a collection
?
Collection maintainers are responsible for releasing new collection versions. The general release process includes:
Planning and announcement
: Define the release scope and communicate it.
Changelog generation
: Create a comprehensive list of changes.
Git tagging
: Create and push a release Git tag.
Automated publication
: The release tarball is automatically published on
Ansible Galaxy
via the
Zuul dashboard
or a custom GitHub Actions workflow.
Final announcement
: Communicate the successful release.
See
Releasing collections
for more information.
Backporting
?
Collection maintainers backport merged pull requests to stable branches if they exist. This process adheres to the collection?s
semantic versioning
and release policies.
The manual backporting process mirrors the
ansible-core backporting guidelines
.
For streamlined backporting, GitHub bots like the
Patchback app
can automate the process through labeling, as implemented in the
community.general
collection.
Including a collection in Ansible
?
To include a collection in the Ansible community package, maintainers create a discussion in the
ansible-collections/ansible-inclusion repository
. See the
submission process
and the
Ansible community package collections requirements
for details.
Stepping down as a collection maintainer
?
If you can no longer continue as a collection maintainer, follow these steps:
Inform other maintainers
: Notify your co-maintainers.
Notify the community
: For collections under the
ansible-collections
GitHub organization, inform the relevant
Real-time chat

channels, or email
ansible-community@redhat.com
.
Identify potential replacements
: Look for active contributors within the collection who could become new maintainers. Discuss these candidates with other maintainers or the
Ansible community team
.
Announce the need for maintainers (if no replacement is found)
: If you cannot find a replacement, create a pinned issue in the collection repository announcing the need for new maintainers.
Post in the Bullhorn newsletter
: Make the same announcement through the
Bullhorn newsletter
.
Engage in candidate discussions
: Be available to discuss potential candidates identified by other maintainers or the community team.
Remember, this is a community and you are welcome to rejoin at any time.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_api.html

Python API ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Python API

Attention

The Ansible API is intended for internal Ansible use. Ansible may make changes to this API at any time that could break backward compatibility with older versions of the API. Because of this, external use is not supported by Ansible. If you want to use Python API only for executing playbooks or modules, consider
ansible-runner
first.

If you would like to use Ansible programmatically from a language other than Python, trigger events asynchronously, or have access control and logging demands, please see the
AWX project
.

See also

Developing dynamic inventory
Developing dynamic inventory integrations
Developing modules
Getting started on developing a module
Developing plugins
How to develop plugins
Communication

Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide

Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_collections.html

Developing collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Developing collections

?

Collections are a distribution format for Ansible content. You can package and distribute playbooks, roles, modules, and plugins using collections. A typical collection addresses a set of related use cases. For example, the
cisco.ios
collection automates management of Cisco IOS devices.

You can create a collection and publish it to
Ansible Galaxy
or to a private Automation Hub instance. You can publish certified collections to the Red Hat Automation Hub, part of the Red Hat Ansible Automation Platform.

Examine the
Ansible collection creator path
to understand how to go from creating a collection to having it included in the Ansible package distribution.

Migrating Ansible content to a different collection
Migrating content
Contributing to collections
Contributing to a collection: community.general
Generating changelogs and porting guide entries in a collection
Understanding antsibull-changelog
Including collection changelogs into Ansible
Collections references
Collection structure
Collection directories and files
Collection Galaxy metadata structure
Structure
Examples
For instructions on developing modules, see
Developing modules
.
See also
Using Ansible collections
Learn how to install and use collections in playbooks and roles
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_creating.h

Creating collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing collections
Creating collections
Edit on GitHub
Creating collections
?
To create a collection:
Create a
new collection
, optionally using a custom
collection template
, with the
ansible-galaxy
collection
init
command.
Add modules and other content to the collection.
Build the collection into a collection artifact with
ansible-galaxy collection build
.
Publish the collection artifact to Galaxy with
ansible-galaxy collection publish
.
A user can then install your collection on their systems.
Naming your collection
Creating a new collection
Creating a collection from a custom template
Creating collections with ansible-creator
Naming your collection
?
Collection names consist of a namespace and a name, separated by a period (
.
). Both namespace and name should be valid Python identifiers. This means that they should consist of ASCII letters, digits, and underscores.
Note
Usually namespaces and names use lower-case letters, digits, and underscores, but no upper-case letters.
You should make sure that the namespace you use is not registered by someone else by checking on
Ansible Galaxy?s namespace list
. If you chose a namespace or even a full collection name that collides with another collection on Galaxy, it can happen

that if you or someone else runs
ansible-galaxy
collection
install
with your collection name, you end up with another collection. Even if the namespace currently does not exist, it could be created later by someone else.
If you want to request a new namespace on Ansible Galaxy,
create an issue on github.com/ansible/galaxy
.
There are a few special namespaces:
ansible
:
The
ansible namespace
is owned by Red Hat and reserved for official Ansible collections. Two special members are the synthetic
ansible.builtin
and
ansible.legacy
collections. These cannot be found on Ansible Galaxy, but are built-in into ansible-core.
community
:
The
community namespace
is owned by the Ansible community. Collections from this namespace generally live in the
GitHub ansible-collection organization
. If you want to create a collection in this namespace,
request
it on the forum.
local
:
The
local namespace
does not contain any collection on Ansible Galaxy, and the intention is that this will never change. You can use the
local
namespace for collections that are locally on your machine or locally in your Git repositories, without having to fear collisions with actually existing collections on Ansible Galaxy.
Creating a new collection
?
Create your collection skeleton in a path that includes
ansible_collections
, for example
collections/ansible_collections/
.
To start a new collection, run the following command in your collections directory:
ansible_collections#>
ansible-galaxy
collection
init
my_namespace.my_collection
Note
Both the namespace and collection names use the same strict set of requirements. Both are limited to alphanumeric characters and underscores, must have a minimum length of two characters, and cannot start with an underscore.
It will create the structure
[my_namespace]/[my_collection]/[collection

skeleton]
.
Hint
If Git is used for version control, the corresponding repository should be initialized in the collection directory.
Once the collection exists, you can populate the directories with the content you want inside the collection. See
ansible-collections
GitHub Org to get a better idea of what you can place inside a collection.
Reference: the
ansible-galaxy
collection
command
Currently the
ansible-galaxy
collection
command implements the following sub commands:
init
: Create a basic collection based on the default template included with Ansible or your own template.
build
: Create a collection artifact that can be uploaded to Galaxy or your own repository.
publish
: Publish a built collection artifact to Galaxy.
install
: Install one or more collections.
To learn more about the
ansible-galaxy
command-line tool, see the
ansible-galaxy
man page.
Creating a collection from a custom template
?
The built-in collection template is a simple example of a collection that works with
ansible-core
, but if you want to simplify your development process you may want to create a custom collection template to pass to
ansible-galaxy
collection
init
.
A collection skeleton is a directory that looks like a collection directory but any
.j2
files (excluding those in
templates/
and
roles/*/templates/
) will be templated by
ansible-galaxy
collection
init
. The skeleton?s
galaxy.yml.j2
file should use the variables
namespace
and
collection_name
which are derived from

ansible-galaxy
init
namespace.collection_name
, and will populate the metadata in the initialized collection?s
galaxy.yml
file. There are a few additional variables available by default (for example,
version
is
1.0.0
), and these can be supplemented/overridden using
--extra-vars
.
An example
galaxy.yml.j2
file that accepts an optional dictionary variable
dependencies
could look like this:

```
namespace:
{{
namespace
}}
name:
{{
collection_name
}}
version:
{{
(
version
|
quote
)
is
version
(
'0.0.0'
,
operator
=
'gt'
,
version_type
=
'semver'
)|
ternary
(
version
,
undef
(
'version must be a valid semantic version greater than 0.0.0'
))
}}
```

dependencies:
{{
dependencies
|
default
({},
true
)
}}
To initialize a collection using the new template, pass the path to the skeleton with
ansible-galaxy
collection
init
:
ansible_collections#>
ansible-galaxy
collection
init
--collection-skeleton
/path/to/my/namespace/skeleton
--extra-vars
"@my_vars_file.json"
my_namespace.my_collection
Note
Before
ansible-core
2.17, collection skeleton templating is limited to the few hardcoded variables including
namespace
,
collection_name
, and
version
.
Note
The default collection skeleton uses an internal filter
comment_ify
that isn?t accessibly to
--collection-skeleton
. Use
ansible-doc
-t
filter|test
--list
to see available plugins.
Creating collections with ansible-creator
?
ansible-creator
is designed to quickly scaffold an Ansible collection project.
Note
The
Ansible Development Tools
package offers a convenient way to install
ansible-creator
along with a curated set of tools for developing automation content.

After
installing
ansible-creator
you can initialize a project in one of the following ways:
Use the
init
subcommand.
Use
ansible-creator
with the
Ansible extension
in Visual Studio Code.
See also
Using Ansible collections
Learn how to install and use collections.
Collection structure
Directories and files included in the collection skeleton
Ansible Development Tools (ADT)
Python package of tools to create and test Ansible content.
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_testing.ht

Testing collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Testing collections
?
Testing your collection ensures that your code works well and integrates well with the rest of the Ansible ecosystem. Your collection should pass the sanity tests for Ansible code. You should also add unit tests to cover the code in your collection and integration tests to cover the interactions between your collection and ansible-core.

Testing tools
?
The main tool for testing collections is
ansible-test
, Ansible?s testing tool described in
Testing Ansible
and provided by both the
ansible
and
ansible-core
packages.
Use
ansible-test
tool from your collection directory, which must include
ansible_collections
in the path, for example
collections/ansible_collections/community/general
for the
community.general
collection. See
Testing Collection Contributions
and
Testing Ansible and Collections
for testing guidelines.
You can run several sanity tests, as well as run unit and integration tests for plugins using
ansible-test

. When you test collections, test against the ansible-core version(s) you are targeting.
You must always execute
ansible-test
from the root directory of a collection.
You can run
ansible-test
in Docker containers without installing any special requirements.
The Ansible team uses this approach in Azure Pipelines both in the ansible/ansible GitHub repository and in the large community collections such as
community.general
and
community.network
to automatically run the tests when pull requests are submitted.
Many collections which do not require running tests on different OS distributions use GitHub Actions as their continuous integration (CI) platform.
The
collection_template repository
contains GitHub Actions workflow
templates
that collection developers are free to use to easily set up CI in their collection repositories.
The examples below demonstrate running tests in Docker containers.
Sanity tests
?
To run all sanity tests:
ansible-test sanity --docker default -v
See
Sanity Tests
for more information. See the
full list of sanity tests
for details on the sanity tests and how to fix identified issues.
Adding unit tests
?
You must place unit tests in the appropriate
tests/unit/plugins/
directory. For example, you would place tests for
plugins/module_utils/foo/bar.py
in
tests/unit/plugins/module_utils/foo/test_bar.py
or
tests/unit/plugins/module_utils/foo/bar/test_bar.py
. For examples, see the
unit tests in community.general
.
To run all unit tests for all supported Python versions:
ansible-test units --docker default -v
To run all unit tests only for a specific Python version:
ansible-test units --docker default -v --python 3.6
To run only a specific unit test:
ansible-test units --docker default -v --python 3.6 tests/unit/plugins/module_utils/foo/test_bar.py
You can specify Python requirements in the
tests/unit/requirements.txt
file. See
Unit Tests
for more information, especially on fixture files.

Adding integration tests
?
You must place integration tests in the appropriate
tests/integration/targets/
directory. For module integration tests, you can use the module name alone. For example, you would place integration tests for
plugins/modules/foo.py
in a directory called
tests/integration/targets/foo/
. For non-module plugin integration tests, you must add the plugin type to the directory name. For example, you would place integration tests for
plugins/connections/bar.py
in a directory called
tests/integration/targets/connection_bar/
. For lookup plugins, the directory must be called
lookup_foo
, for inventory plugins,
inventory_foo
, and so on.
You can write two different kinds of integration tests:
Ansible role tests run with
ansible-playbook
and validate various aspects of the module. They can depend on other integration tests (usually named
prepare_bar
or
setup_bar
, which prepare a service or install a requirement named
bar
in order to test module
foo
) to set-up required resources, such as installing required libraries or setting up server services.
runme.sh
tests run directly as scripts. They can set up inventory files, and execute
ansible-playbook
or
ansible-inventory
with various settings.
For examples, see the
integration tests in community.general
. See also
Integration tests
for more details.
Since integration tests can install requirements, and set-up, start and stop services, we recommended running them in docker containers or otherwise restricted environments whenever possible. By default,
ansible-test
supports Docker images for several operating systems. See the
list of supported docker images
for all options. Use the
default
image mainly for platform-independent integration tests, such as those for cloud modules. The following examples use the
fedora35
image.
To execute all integration tests for a collection:

ansible-test integration --docker fedora35 -v
If you want more detailed output, run the command with
-vvv
instead of
-v
. Alternatively, specify
--retry-on-error
to automatically re-run failed tests with higher verbosity levels.
To execute only the integration tests in a specific directory:
ansible-test integration --docker fedora35 -v connection_bar
You can specify multiple target names. Each target name is the name of a directory in
tests/integration/targets/
.
See also
Testing Ansible
More resources on testing Ansible
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_locally.html

Adding modules and plugins locally ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Edit on GitHub
Adding modules and plugins locally
?
You can extend Ansible by adding custom modules or plugins. You can create them from scratch or copy existing ones for local use. You can store a local module or plugin on your Ansible control node and share it with your team or organization. You can also share plugins and modules by including them in a collection, then publishing the collection on Ansible Galaxy.
If you are using a local module or plugin but Ansible cannot find it, this page is all you need.
If you want to create a plugin or a module, see
Developing plugins
,
Developing modules
and
Developing collections
.
Extending Ansible with local modules and plugins offers shortcuts such as:
You can copy other people?s modules and plugins.
When writing a new module, you can choose any programming language you like.
You do not have to clone any repositories.
You do not have to open a pull request.
You do not have to add tests (though we recommend that you do!).

Modules and plugins: what is the difference?
?
If you are looking to add functionality to Ansible, you might wonder whether you need a module or a plugin. Here is a

quick overview to help you understand what you need:
Plugins
extend Ansible?s core functionality. Most plugin types execute on the control node within the
/usr/bin/ansible
process. Plugins offer options and extensions for the core features of Ansible: transforming data, logging output, connecting to inventory, and more.
Modules are a type of plugin that execute automation tasks on a ?target? (usually a remote system). Modules work as standalone scripts that Ansible executes in their own process outside of the control node. Modules interface with Ansible mostly with JSON, accepting arguments and returning information by printing a JSON string to stdout before exiting. Unlike the other plugins (which must be written in Python), modules can be written in any language; although Ansible provides modules in Python and Powershell only.
Adding modules and plugins in collections
?
You can add modules and plugins by
creating a collection
. With a collection, you can use custom modules and plugins in any playbook or role. You can share your collection easily at any time through Ansible Galaxy.
The rest of this page describes other methods of using local, standalone modules or plugins.
Adding a module or plugin outside of a collection
?
You can configure Ansible to load standalone local modules or plugins in specific locations and make them available to all playbooks and roles (using configured paths). Alternatively, you can make a non-collection local module or plugin available only to certain playbooks or roles (with adjacent paths).
Adding standalone local modules for all playbooks and roles
?
To load standalone local modules automatically and make them available to all playbooks and roles, use the
DEFAULT_MODULE_PATH
configuration setting or the
ANSIBLE_LIBRARY
environment variable. The configuration setting and environment variable take a colon-separated list, similar to
$PATH
. You have two options:
Add your standalone local module to one of the default configured locations. See the
DEFAULT_MODULE_PATH
configuration setting for details. Default locations may change without notice.
Add the location of your standalone local module to an environment variable or configuration:
the
ANSIBLE_LIBRARY
environment variable
the
DEFAULT_MODULE_PATH
configuration setting
To view your current configuration settings for modules:
ansible-config dump |grep DEFAULT_MODULE_PATH
After you save your module file in one of these locations, Ansible loads it and you can use it in any local task, playbook, or role.
To confirm that
my_local_module
is available:
type
ansible
localhost
-m
my_local_module

to see the output for that module, or
type
ansible-doc
-t
module
my_local_module
to see the documentation for that module
Note
This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.
Note
The
ansible-doc
command can parse module documentation from modules written in Python or an adjacent YAML file. If you have a module written in a programming language other than Python, you should write the documentation in a Python or YAML file adjacent to the module file.
Adjacent YAML documentation files
Adding standalone local modules for selected playbooks or a single role
?
Ansible automatically loads all executable files from certain directories adjacent to your playbook or role as modules. Standalone modules in these locations are available only to the specific playbook, playbooks, or role in the parent directory.
To use a standalone module only in a selected playbook or playbooks, store the module in a subdirectory called
library
in the directory that contains the playbook or playbooks.
To use a standalone module only in a single role, store the module in a subdirectory called
library
within that role.
Note
This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.
Warning
Roles contained in collections cannot contain any modules or other plugins. All plugins in a collection must live in the collection
plugins
directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new modules, we recommend distributing them in
collections
, not in roles.
Adding a non-module plugin locally outside of a collection
?
You can configure Ansible to load standalone local plugins in a specified location or locations and make them available to all playbooks and roles. Alternatively, you can make a standalone local plugin available only to specific playbooks or roles.
Note
Although modules are plugins, the naming patterns for directory names and environment variables that apply to other plugin types do not apply to modules. See
Adding a module or plugin outside of a collection
.
Adding local non-module plugins for all playbooks and roles
?
To load standalone local plugins automatically and make them available to all playbooks and roles, use the configuration setting or environment variable for the type of plugin you are adding. These configuration settings and environment variables take a colon-separated list, similar to

$PATH
. You have two options:
Add your local plugin to one of the default configured locations. See
configuration settings
for details on the correct configuration setting for the plugin type. Default locations may change without notice.
Add the location of your local plugin to an environment variable or configuration:
the relevant
ANSIBLE_plugin_type_PLUGINS
environment variable - for example,
$ANSIBLE_INVENTORY_PLUGINS
or
$ANSIBLE_VARS_PLUGINS
the relevant
plugin_type_PATH
configuration setting, most of which begin with
DEFAULT_
- for example,
DEFAULT_CALLBACK_PLUGIN_PATH
or
DEFAULT_FILTER_PLUGIN_PATH
or
BECOME_PLUGIN_PATH
To view your current configuration settings for non-module plugins:
ansible-config dump |grep plugin_type_PATH
After your plugin file is added to one of these locations, Ansible loads it and you can use it in any local module, task, playbook, or role. For more information on environment variables and configuration settings, see
Ansible Configuration Settings
.
To confirm that
plugins/plugin_type/my_local_plugin
is available:
type
ansible-doc
-t
<plugin_type>
my_local_lookup_plugin
to see the documentation for that plugin - for example,
ansible-doc
-t
lookup
my_local_lookup_plugin
The
ansible-doc
command works for most plugin types, but not for action, filter, or test plugins. See
ansible-doc
for more details.
Adding standalone local plugins for selected playbooks or a single role
?
Ansible automatically loads all plugins from certain directories adjacent to your playbook or role, loading each type of plugin separately from a directory named for the type of plugin. Standalone plugins in these locations are available only to the specific playbook, playbooks, or role in the parent directory.
To use a standalone plugin only in a selected playbook or playbooks, store the plugin in a subdirectory for the correct
plugin_type
(for example,

callback_plugins
or
inventory_plugins
) in the directory that contains the playbooks. These directories must use the
_plugins
suffix. For a full list of plugin types, see
Working with plugins
.
To use a standalone plugin only in a single role, store the plugin in a subdirectory for the correct
plugin_type
(for example,
cache_plugins
or
strategy_plugins
) within that role. When shipped as part of a role, the plugin is available as soon as the role is executed. These directories must use the
_plugins
suffix. For a full list of plugin types, see
Working with plugins
.
Warning
Roles contained in collections cannot contain any plugins. All plugins in a collection must live in the collection
plugins
directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new plugins, we recommend distributing them in
collections
, not in roles.
Warning
Some plugin types are needed early during Ansible execution, such as callbacks, inventory, and cache. These plugin types cannot be loaded dynamically and must exist in configured paths or be referenced by FQCN in configuration.
Using
ansible.legacy
to access custom versions of an
ansible.builtin
module
?
If you need to override one of the
ansible.builtin
modules and are using FQCN, you need to use
ansible.legacy
as part of the fully-qualified collection name (FQCN). For example, if you had your own
copy
module, you would access it as
ansible.legacy.copy
. See
Using ansible.legacy to access local custom modules from collections-based roles
for details on how to use custom modules with roles within a collection.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_checklist.htr

Contributing your module to an existing Ansible collection ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Edit on GitHub
Contributing your module to an existing Ansible collection
?
If you want to contribute a module to an existing collection, you must meet the community?s objective and subjective requirements. Please read the details below, and also review our
tips for module development
.
Modules accepted into certain collections are included in every Ansible release on PyPI. However, contributing to one of these collections is not the only way to distribute a module - you can
create your own collection
, embed modules in roles on Galaxy or simply share copies of your module code for
local use
.
Contributing modules: objective requirements
?
To contribute a module to most Ansible collections, you must:
write your module in either Python or Powershell for Windows
use the
AnsibleModule
common code
support Python 2.6 and Python 3.5 - if your module cannot support Python 2.6, explain the required minimum Python version and rationale in the requirements section in
DOCUMENTATION
use proper
Python 3 syntax
follow
PEP 8
Python style conventions - see
pep8
for more information
license your module under the GPL license (GPLv3 or later)
understand the
DCO agreement
, which applies to contributions to the
Ansible Core
and
Ansible Documentation

repositories.
conform to Ansible?s
formatting and documentation
standards
include comprehensive
tests
for your module
minimize module dependencies
support
check_mode
if possible
ensure your code is readable
if a module is named
<something>_facts
, it should be because its main purpose is returning
ansible_facts
. Do not name modules that do not do this with
_facts
. Only use
ansible_facts
for information that is specific to the host machine, for example network interfaces and their configuration, which operating system and which programs are installed.
Modules that query/return general information (and not
ansible_facts
) should be named
_info
. General information is non-host specific information, for example information on online/cloud services (you can access different accounts for the same online service from the same host), or information on VMs and containers accessible from the machine.
Additional requirements may apply for certain collections. Review the individual collection repositories for more information.
Please make sure your module meets these requirements before you submit your PR/proposal. If you have questions, visit the
Ansible communication guide
for information on how to reach out to the community.
Contributing to Ansible: subjective requirements
?
If your module meets these objective requirements, collection maintainers will review your code to see if they think it is clear, concise, secure, and maintainable. They will consider whether your module provides a good user experience, helpful error messages, reasonable defaults, and more. This process is subjective, with no exact standards for acceptance. For the best chance of getting your module accepted, follow our
tips for module development
.
Other checklists
?
Tips for module development
.
Windows development checklist
.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general.html

Developing modules ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Developing modules
?
A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. Each module can be used by the Ansible API, or by the
ansible
or
ansible-playbook
programs. A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.
If you need functionality that is not available in any of the thousands of Ansible modules found in collections, you can easily write your own custom module. When you write a module for local use, you can choose any programming language and follow your own rules. Use this topic to learn how to create an Ansible module in Python. After you create a module, you must add it locally to the appropriate directory so that Ansible can find and execute it. For details about adding a module locally, see
Adding modules and plugins locally
.
If you are developing a module in a
collection
, see those documents instead.

Preparing an environment for developing Ansible modules
?
You just need
ansible-core
installed to test the module. Modules can be written in any language,

but most of the following guide is assuming you are using Python.
Modules for inclusion in Ansible itself must be Python or Powershell.
One advantage of using Python or Powershell for your custom modules is being able to use the
module_utils
common code that does a lot of the
heavy lifting for argument processing, logging and response writing, among other things.
Creating a standalone module
?
It is highly recommended that you use a
venv
or
virtualenv
for Python development.
To create a standalone module:
Create a
library
directory in your workspace. Your test play should live in the same directory.
Create your new module file:
$
touch
library/my_test.py
. Or just open/create it with your editor of choice.
Paste the content below into your new module file. It includes the
required Ansible format and documentation
, a simple
argument spec for declaring the module options
, and some example code.
Modify and extend the code to do what you want your new module to do. See the
programming tips
and
Python 3 compatibility
pages for pointers on writing clean and concise module code.
Creating a module in a collection
?
To create a new module in an existing collection called
my_namespace.my_collection
:
Create your new module file:
$
touch
<PATH_TO_COLLECTION>/ansible_collections/my_namespace/my_collection/plugins/modules/my_test.py
. Or just create it with your editor of choice.
Paste the content below into your new module file. It includes the
required Ansible format and documentation
, a simple
argument spec for declaring the module options
, and some example code.
Modify and extend the code to do what you want your new module to do. See the
programming tips
and
Python 3 compatibility
pages for pointers on writing clean and concise module code.
#!/usr/bin/python
Copyright: (c) 2018, Terry Jones <

```
[email protected]
>
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.txt)
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
DOCUMENTATION
=
r
'''
---
module: my_test
short_description: This is my test module
# If this is part of a collection, you need to use semantic versioning,
# i.e. the version is of the form "2.5.0" and not "2.4".
version_added: "1.0.0"
description: This is my longer description explaining my test module.
options:
name:
description: This is the message to send to the test module.
required: true
type: str
new:
description:
- Control to demo if the result of this module is changed or not.
- Parameter description can be a list as well.
required: false
type: bool
# Specify this value according to your collection
# in format of namespace.collection.doc_fragment_name
# extends_documentation_fragment:
#     - my_namespace.my_collection.my_doc_fragment_name
author:
- Your Name (@yourGitHubHandle)
'''
EXAMPLES
=
r
'''
# Pass in a message
- name: Test with a message
my_namespace.my_collection.my_test:
name: hello world
# pass in a message and have changed true
```

```
- name: Test with a message and changed output
my_namespace.my_collection.my_test:
name: hello world
new: true
# fail the module
- name: Test failure of the module
my_namespace.my_collection.my_test:
name: fail me
'''
RETURN
=
r
'''
# These are examples of possible return values, and in general should use other names for return values.
original_message:
description: The original name param that was passed in.
type: str
returned: always
sample: 'hello world'
message:
description: The output message that the test module generates.
type: str
returned: always
sample: 'goodbye'
'''
from
ansible.module_utils.basic
import
AnsibleModule
def
run_module
():
# define available arguments/parameters a user can pass to the module
module_args
=
dict
(
name
=
dict
(
type
=
'str'
,
required
=
True
),
new
=
dict
(
type
```

=
'bool'
,
required
=
False
,
default
=
False
)
)
seed the result dict in the object
we primarily care about changed and state
changed is if this module effectively modified the target
state will include any data that you want your module to pass back
for consumption, for example, in a subsequent task
result
=
dict
(
changed
=
False
,
original_message
=
''
,
message
=
''
)
the AnsibleModule object will be our abstraction working with Ansible
this includes instantiation, a couple of common attr would be the
args/params passed to the execution, as well as if the module
supports check mode
module
=
AnsibleModule
(
argument_spec
=
module_args
,
supports_check_mode
=
True
)
if the user is working with this module in only check mode we do not
want to make any changes to the environment, just return the current
state with no modifications
if
module

```
.
check_mode
:
module
.
exit_json
(
**
result
)
# manipulate or modify the state as needed (this is going to be the
# part where your module will do what it needs to do)
result
[
'original_message'
]
=
module
.
params
[
'name'
]
result
[
'message'
]
=
'goodbye'
# use whatever logic you need to determine whether or not this module
# made any modifications to your target
if
module
.
params
[
'new'
]:
result
[
'changed'
]
=
True
# during the execution of the module, if there is an exception or a
# conditional state that effectively causes a failure, run
# AnsibleModule.fail_json() to pass in the message and the result
if
module
.
params
[
'name'
]
```

==
'fail me'
:
module
.
fail_json
(
msg
=
'You requested this to fail'
,
**
result
)
in the event of a successful module execution, you will want to
simple AnsibleModule.exit_json(), passing the key/value results
module
.
exit_json
(
**
result
)
def
main
():
run_module
()
if
__name__
==
'__main__'
:
main
()
Creating an info or a facts module
?
Ansible gathers information about the target machines using facts modules, and gathers information on other objects or files using info modules.
If you find yourself trying to add
state:
info
or
state:
list
to an existing module, that is often a sign that a new dedicated
_facts
or
_info
module is needed.
In Ansible 2.8 and onwards, we have two type of information modules, they are
*_info
and
*_facts

.
If a module is named
<something>_facts
, it should be because its main purpose is returning
ansible_facts
. Do not name modules that do not do this with
_facts
.
Only use
ansible_facts
for information that is specific to the host machine, for example network interfaces and their configuration, which operating system and which programs are installed.
Modules that query/return general information (and not
ansible_facts
) should be named
_info
.
General information is non-host specific information, for example information on online/cloud services (you can access different accounts for the same online service from the same host), or information on VMs and containers accessible from the machine, or information on individual files or programs.
Info and facts modules, are just like any other Ansible Module, with a few minor requirements:
They MUST be named
<something>_info
or
<something>_facts
, where <something> is singular.
Info
*_info
modules MUST return in the form of the
result dictionary
so other modules can access them.
Fact
*_facts
modules MUST return in the
ansible_facts
field of the
result dictionary
so other modules can access them.
They MUST support
check_mode
.
They MUST NOT make any changes to the system.
They MUST document the
return fields
and
examples
.
You can add your facts into
ansible_facts
field of the result as follows:
module
.
exit_json
(

changed
=
False
,
ansible_facts
=
dict
(
my_new_fact
=
value_of_fact
))
The rest is just like creating a normal module.
Verifying your module code
?
After you modify the sample code above to do what you want, you can try out your module.
Our
debugging tips
will help if you run into bugs as you verify your module code.
Verifying your module code locally
?
The simplest way is to use
ansible
adhoc command:
ANSIBLE_LIBRARY
=
./library
ansible
-m
my_test
-a
'name=hello new=true'
remotehost
If your module does not need to target a remote host, you can quickly and easily exercise your code locally like this:
ANSIBLE_LIBRARY
=
./library
ansible
-m
my_test
-a
'name=hello new=true'
localhost
For a module developed in an existing collection called
my_namespace.my_collection
(as mentioned above):
$
ansible
localhost
-m
my_namespace.my_collection.my_test
-a
'name=hello new=true'
--playbook-dir

=
$PWD
If you use
pdb
,
print()
, or some other method of local debugging for faster iteration,
you can avoid going through Ansible by creating an arguments file, which is
a basic JSON config file that passes parameters to your module so that you can run it.
Name the arguments file
/tmp/args.json
and add the following content:
{
"ANSIBLE_MODULE_ARGS"
:
{
"name"
:
"hello"
,
"new"
:
true
}
}
Then the module can be tested locally and directly. This skips the packing steps and uses module_utils files directly:
$
python
library/my_test.py
/tmp/args.json
You might also need to add your collection?s path to the Python path. This tells Python to look for additional module_utils code in the given path.
You can run the module code, as in the following example:
$
export
PYTHONPATH
=
PATH_TO_COLLECTIONS:
$PYTHONPATH
$
python
-m
ansible_collections.my_namespace.my_collection.plugins.modules.my_test
/tmp/args.json
It should return output like this:
{
"changed"
:
true
,
"state"
:
{
"original_message"

:
"hello"
,
"new_message"
:
"goodbye"
},
"invocation"
:
{
"module_args"
:
{
"name"
:
"hello"
,
"new"
:
true
}}}
Verifying your module code in a playbook
?
You can easily run a full test by including it in a playbook, as long as the
library
directory is in the same directory as the play:
Create a playbook in any directory:
$
touch
testmod.yml
Add the following to the new playbook file:
-
name
:
test my new module
hosts
:
localhost
tasks
:
-
name
:
run the new module
my_test
:
name
:
'hello'
new
:
true
register
:

testout
-
name
:
dump test output
debug
:
msg
:
'{{
testout
}}'
Run the playbook and analyze the output:
$
ansible-playbook
./testmod.yml
Testing your newly-created module
?
Review our
testing
section for more detailed
information, including instructions for
testing module documentation
, adding
integration tests
, and more.
Note
If contributing to Ansible, every new module and plugin should have integration tests, even if the tests cannot be run on Ansible CI infrastructure.
In this case, the tests should be marked with the
unsupported
alias in
aliases file
.
Contributing back to Ansible
?
If you would like to contribute to
ansible-core
by adding a new feature or fixing a bug,
create a fork
of the ansible/ansible repository and develop against a new feature branch using the
devel
branch as a starting point. When you have a good working code change, you can submit a pull request to the Ansible repository by selecting your feature branch as a source and the Ansible devel branch as a target.
If you want to contribute a module to an
Ansible collection
, review our
submission checklist
,
programming tips
, and
strategy for maintaining Python 2 and Python 3 compatibility
, as well as information about
testing

before you open a pull request.
The
Community Guide
covers how to open a pull request and what happens next.
Communication and development support
?
Visit the
Ansible communication guide
for information on how to join the conversation.
Credit
?
Thank you to Thomas Stringer (
@trstringer
) for contributing source
material for this topic.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/developing_python_3.html

Ansible and Python 3 ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Ansible and Python 3
Edit on GitHub
Ansible and Python 3
?
The
ansible-core
code runs Python 3 (for specific versions check
Control Node Requirements
Contributors to
ansible-core
and to Ansible Collections should be aware of the tips in this document so that they can write code that will run on the same versions of Python as the rest of Ansible.
Minimum version of Python 3.x and Python 2.x
Developing Ansible code that supports Python 2 and Python 3
Understanding strings in Python 2 and Python 3
Control node string strategy: the Unicode Sandwich
Unicode Sandwich common borders: places to convert bytes to text in control node code
Reading and writing to files
Filesystem interaction
Interacting with other programs
Module string strategy: Native String
Module_utils string strategy: hybrid
Tips, tricks, and idioms for Python 2/Python 3 compatibility
Use forward-compatibility boilerplate
Prefix byte strings with
b_
Import Ansible?s bundled Python
six
library
Handle exceptions with
as
Update octal numbers
String formatting for control node code
Use
str.format()
for Python 2.6 compatibility
Use percent format with byte strings
We do have some considerations depending on the types of Ansible code:

code on the control node - code that runs on the machine where you invoke
/usr/bin/ansible
, only needs to support the control node?s Python versions.
modules - the code which Ansible transmits to and invokes on the managed machine. Modules need to support the ?managed node? Python versions, with some exceptions.
shared
module_utils
code - the common code that is used by modules to perform tasks and sometimes used by code on the control node.
Shared
module_utils
code needs to support the same range of Python as the modules.
However, the three types of code do not use the same string strategy. If you?re developing a module or some
module_utils
code, be sure to read the section on string strategy carefully.
Minimum version of Python 3.x and Python 2.x
?
See
Control Node Requirements
and
Managed Node Requirements
for the
specific versions supported.
Your custom modules can support any version of Python (or other languages) you want, but the above are the requirements for the code contributed to the Ansible project.
Developing Ansible code that supports Python 2 and Python 3
?
The best place to start learning about writing code that supports both Python 2 and Python 3
is
Lennart Regebro?s book: Porting to Python 3
.
The book describes several strategies for porting to Python 3. The one we?re
using is
to support Python 2 and Python 3 from a single code base
Understanding strings in Python 2 and Python 3
?
Python 2 and Python 3 handle strings differently, so when you write code that supports Python 3
you must decide what string model to use. Strings can be an array of bytes (like in C) or
they can be an array of text. Text is what we think of as letters, digits,
numbers, other printable symbols, and a small number of unprintable ?symbols?
(control codes).
In Python 2, the two types for these (
str
for bytes and
unicode
for text) are often used interchangeably. When dealing only
with ASCII characters, the strings can be combined, compared, and converted
from one type to another automatically. When non-ASCII characters are
introduced, Python 2 starts throwing exceptions due to not knowing what encoding
the non-ASCII characters should be in.
Python 3 changes this behavior by making the separation between bytes (
bytes
)
and text (
str

) more strict. Python 3 will throw an exception when
trying to combine and compare the two types. The programmer has to explicitly
convert from one type to the other to mix values from each.
In Python 3 it is immediately apparent to the programmer when code is
mixing the byte and text types inappropriately, whereas in Python 2, code that mixes those types
may work until a user causes an exception by entering non-ASCII input.
Python 3 forces programmers to proactively define a strategy for
working with strings in their program so that they don?t mix text and byte strings unintentionally.
Ansible uses different strategies for working with strings in the code on the control node, in
:ref:
modules <module_string_strategy>
, and in
module_utils
code.
Control node string strategy: the Unicode Sandwich
?
Until recently
ansible-core
supported Python 2.x and followed this strategy, known as the Unicode Sandwich (named
after Python 2?s
unicode
text type). For Unicode Sandwich we know that
at the border of our code and the outside world (for example, file and network IO,
environment variables, and some library calls) we are going to receive bytes.
We need to transform these bytes into text and use that throughout the
internal portions of our code. When we have to send those strings back out to
the outside world we first convert the text back into bytes.
To visualize this, imagine a ?sandwich? consisting of a top and bottom layer
of bytes, a layer of conversion between, and all text type in the center.
For compatibility reasons you will see a bunch of custom functions we developed (
to_text
/
to_bytes
/
to_native
)
and while Python 2 is not a concern anymore we will continue to use them as they apply for other cases that make
dealing with unicode problematic.
While we will not be using it most of it anymore, the documentation below is still useful for those developing modules
that still need to support both Python 2 and 3 simultaneously.
Unicode Sandwich common borders: places to convert bytes to text in control node code
?
This is a partial list of places where we have to convert to and from bytes
when using the Unicode Sandwich string strategy. It is not exhaustive but
it gives you an idea of where to watch for problems.
Reading and writing to files
?
In Python 2, reading from files yields bytes. In Python 3, it can yield text.
To make code that?s portable to both we don?t make use of Python 3?s ability
to yield text but instead do the conversion explicitly ourselves. For example:
from
ansible.module_utils.common.text.converters
import
to_text

```
with
open
(
'filename-with-utf8-data.txt'
,
'rb'
)
as
my_file
:
b_data
=
my_file
.
read
()
try
:
data
=
to_text
(
b_data
,
errors
=
'surrogate_or_strict'
)
except
UnicodeError
:
# Handle the exception gracefully -- usually by displaying a good
# user-centric error message that can be traced back to this piece
# of code.
pass
```

Note

Much of Ansible assumes that all encoded text is UTF-8. At some point, if there is demand for other encodings we may change that, but for now it is safe to assume that bytes are UTF-8.

Writing to files is the opposite process:

```
from
ansible.module_utils.common.text.converters
import
to_bytes
with
open
(
'filename.txt'
,
'wb'
)
as
my_file
:
```

my_file
.
write
(
to_bytes
(
some_text_string
))
Note that we don?t have to catch
UnicodeError
here because we?re
transforming to UTF-8 and all text strings in Python can be transformed back
to UTF-8.
Filesystem interaction
?
Dealing with file names often involves dropping back to bytes because on UNIX-like
systems file names are bytes. On Python 2, if we pass a text string to these
functions, the text string will be converted to a byte string inside of the
function and a traceback will occur if non-ASCII characters are present. In
Python 3, a traceback will only occur if the text string can?t be decoded in
the current locale, but it is still good to be explicit and have code which
works on both versions:
import
os.path
from
ansible.module_utils.common.text.converters
import
to_bytes
filename
=
u
'/var/tmp/????.txt'
f
=
open
(
to_bytes
(
filename
),
'wb'
)
mtime
=
os
.
path
.
getmtime
(
to_bytes
(
filename
))

```
b_filename
=
os
.
path
.
expandvars
(
to_bytes
(
filename
))
if
os
.
path
.
exists
(
to_bytes
(
filename
)):
pass
```

When you are only manipulating a filename as a string without talking to the filesystem (or a C library which talks to the filesystem) you can often get away without converting to bytes:

```
import
os.path
os
.
path
.
join
(
u
'/var/tmp/café'
,
u
'????'
)
os
.
path
.
split
(
u
'/var/tmp/café/????'
)
```

On the other hand, if the code needs to manipulate the file name and also talk to the filesystem, it can be more convenient to transform to bytes right away and manipulate in bytes.

Warning

Make sure all variables passed to a function are the same type.
If you?re working with something like
os.path.join()
which takes
multiple strings and uses them in combination, you need to make sure that
all the types are the same (either all bytes or all text). Mixing
bytes and text will cause tracebacks.
Interacting with other programs
?
Interacting with other programs goes through the operating system and
C libraries and operates on things that the UNIX kernel defines. These
interfaces are all byte-oriented so the Python interface is byte oriented as
well. On both Python 2 and Python 3, byte strings should be given to Python?s
subprocess library and byte strings should be expected back from it.
One of the main places in Ansible?s control node code that we interact with
other programs is the connection plugins?
exec_command
methods. These
methods transform any text strings they receive in the command (and arguments
to the command) to execute into bytes and return stdout and stderr as byte strings
Higher level functions (like action plugins?
_low_level_execute_command
)
transform the output into text strings.
Module string strategy: Native String
?
In modules we use a strategy known as Native Strings. This makes things
easier on the community members who maintain so many of Ansible?s
modules, by not breaking backwards compatibility by
mandating that all strings inside of modules are text and converting between
text and bytes at the borders.
Native strings refer to the type that Python uses when you specify a bare
string literal:
"This is a native string"
In Python 2, these are byte strings. In Python 3 these are text strings. Modules should be
coded to expect bytes on Python 2 and text on Python 3.
Module_utils string strategy: hybrid
?
In
module_utils
code we use a hybrid string strategy. Although Ansible?s
module_utils
code is largely like module code, some pieces of it are
used by the control node as well. So it needs to be compatible with modules
and with the control node?s assumptions, particularly the string strategy.
The module_utils code attempts to accept native strings as input
to its functions and emit native strings as their output.
In
module_utils
code:
Functions
must
accept string parameters as either text strings or byte strings.
Functions may return either the same type of string as they were given or the native string type for the Python version

they are run on.
Functions that return strings
must
document whether they return strings of the same type as they were given or native strings.
Module-utils functions are therefore often very defensive in nature.
They convert their string parameters into text (using
ansible.module_utils.common.text.converters.to_text
)
at the beginning of the function, do their work, and then convert
the return values into the native string type (using
ansible.module_utils.common.text.converters.to_native
)
or back to the string type that their parameters received.
Tips, tricks, and idioms for Python 2/Python 3 compatibility
?
Use forward-compatibility boilerplate
?
Use the following boilerplate code at the top of all python files
to make certain constructs act the same way on Python 2 and Python 3:
Make coding more python3-ish
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
__metaclass__
=
type
makes all classes defined in the file into new-style
classes without explicitly inheriting from
object
.
The
__future__
imports do the following:
absolute_import
:
Makes imports look in
sys.path
for the modules being
imported, skipping the directory in which the module doing the importing
lives. If the code wants to use the directory in which the module doing
the importing, there?s a new dot notation to do so.
division
:
Makes division of integers always return a float. If you need to

find the quotient use
x
//
y
instead of
x
/
y
.
print_function
:
Changes
print
from a keyword into a function.
See also
PEP 0328: Absolute Imports
PEP 0238: Division
PEP 3105: Print function
Prefix byte strings with
b_
?
Since mixing text and bytes types leads to tracebacks we want to be clear
about what variables hold text and what variables hold bytes. We do this by
prefixing any variable holding bytes with
b_
. For example:
filename
=
u
'/var/tmp/café.txt'
b_filename
=
to_bytes
(
filename
)
with
open
(
b_filename
)
as
f
:
data
=
f
.
read
()
We do not prefix the text strings instead because we only operate
on byte strings at the borders, so there are fewer variables that need bytes
than text.
Import Ansible?s bundled Python

six
library
?
The third-party Python
six
library exists
to help projects create code that runs on both Python 2 and Python 3. Ansible includes a version of the library in module_utils so that other modules can use it without requiring that it is installed on the remote system. To make use of it, import it like this:
from
ansible.module_utils
import
six
Note
Ansible can also use a system copy of six
Ansible will use a system copy of six if the system copy is a later version than the one Ansible bundles.
Handle exceptions with
as
?
In order for code to function on Python 2.6+ and Python 3, use the new exception-catching syntax which uses the
as
keyword:
try
:
a
=
2
/
0
except
ValueError
as
e
:
module
.
fail_json
(
msg
=
"Tried to divide by zero:
%s
"
%
e
)
Do
not
use the following syntax as it will fail on every version of Python 3:
```
try:
    a = 2/0
```

except ValueError, e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
Update octal numbers
?
In Python 2.x, octal literals could be specified as
0755
. In Python 3,
octals must be specified as
0o755
.
String formatting for control node code
?
Use
str.format()
for Python 2.6 compatibility
?
Starting in Python 2.6, strings gained a method called
format()
to put
strings together. However, one commonly used feature of
format()
wasn?t
added until Python 2.7, so you need to remember not to use it in Ansible code:
Does not work in Python 2.6!
new_string
=
"Dear
{}
, Welcome to
{}
"
.
format
(
username
,
location
)
Use this instead
new_string
=
"Dear
{0}
, Welcome to
{1}
"
.
format
(
username
,
location
)
Both of the format strings above map positional arguments of the

format()
method into the string. However, the first version doesn?t work in
Python 2.6. Always remember to put numbers into the placeholders so the code
is compatible with Python 2.6.
See also
Python documentation on format strings:
format strings in 2.6
format strings in 3.x
Use percent format with byte strings
?
In Python 3.5 and later, byte strings do not have a
format()
method. However, it
does have support for the older, percent-formatting.
b_command_line
=
b
'ansible-playbook --become-user
%s
-K
%s
'
%
(
user
,
playbook_file
)
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/module_lifecycle.html

The lifecycle of an Ansible module or plugin ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

The lifecycle of an Ansible module or plugin

?

Modules and plugins in the main Ansible repository have a defined life cycle, from the first introduction to final removal. The module and plugin lifecycle is tied to the
Ansible release cycle <release_cycle>
.

A module or plugin may move through these four stages:

When a module or plugin is first accepted into Ansible, we consider it in tech preview and will mark it as such in the documentation.

If a module or plugin matures, the ?preview? mark in the documentation is removed. Backward compatibility for these modules and plugins is maintained but not guaranteed, which means their parameters should be maintained with stable meanings.

If a module?s or plugin?s target API changes radically, or if someone creates a better implementation of its functionality, we may mark it deprecated. Modules and plugins that are deprecated are still available but they are reaching the end of their life cycle. We retain deprecated modules and plugins for 4 release cycles with deprecation warnings to help users update playbooks and roles that use them.

When a module or plugin has been deprecated for four release cycles, it is removed and replaced with a tombstone entry in the routing configuration. Modules and plugins that are removed are no longer shipped with Ansible. The tombstone entry helps users find alternative modules and plugins.

For modules and plugins in collections, the lifecycle is similar. Since ansible-base 2.10, it is no longer possible to mark modules as ?preview? or ?stable?.

Deprecating modules and plugins in the Ansible main repository

?

To deprecate a module in ansible-core, you must:

Rename the file so it starts with an
_
, for example, rename
old_cloud.py
to
_old_cloud.py
. This keeps the module available and marks it as deprecated on the module index pages.

Mention the deprecation in the relevant changelog (by creating a changelog fragment with a section
deprecated_features
).

Reference the deprecation in the relevant
porting_guide_core_x.y.rst

.
Add
deprecated:
to the documentation with the following sub-values:
removed_in
:
A
string
, such as
"2.10"
; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.
remove_by_date
:
(Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.
why
:
Optional string that used to detail why this has been removed.
alternatives
:
Inform users they should do instead, for example,
Use
M(whatmoduletouseinstead)
instead.
.
For an example of documenting deprecation, see this
PR that deprecates multiple modules
.
Some of the elements in the PR might now be out of date.
Deprecating modules and plugins in a collection
?
To deprecate a module in a collection, you must:
Add a
deprecation
entry to
plugin_routing
in
meta/runtime.yml
. For example, to deprecate the module
old_cloud
, add:
plugin_routing
:
modules
:
old_cloud
:
deprecation
:
removal_version
:
2.0.0
warning_text

:
Use foo.bar.new_cloud instead.
For other plugin types, you have to replace
modules:
with
<plugin_type>:
, for example
lookup:
for lookup plugins. When deprecating action plugins, you need to add two entries: one for the action plugin and one for the module file that contains the documentation.
Instead of
removal_version
, you can also use
removal_date
with an ISO 8601 formatted date after which the module will be removed in a new major version of the collection.
Mention the deprecation in the relevant changelog. If the collection uses
antsibull-changelog
, create a changelog fragment with a section
deprecated_features
.
Add
deprecated:
to the documentation of the module or plugin with the following sub-values:
removed_in
:
A
string
, such as
"2.10"
; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.
remove_by_date
:
(Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.
why
:
String that used to detail why this has been removed.
alternative
:
Inform users they should do instead, for example,
Use
M(whatmoduletouseinstead)
instead.
. See
Linking within module documentation
for ways to reference entities other than modules.
Changing a module or plugin name in the Ansible main repository
?
You can also rename a module and keep a deprecated alias to the old name by using a symlink that starts with _.
This example allows the
stat
module to be called with
fileinfo

, making the following examples equivalent:
ln -s stat.py _fileinfo.py
ansible -m stat -a "path=/tmp" localhost
ansible -m fileinfo -a "path=/tmp" localhost
Renaming a module or plugin in a collection, or redirecting a module or plugin to another collection
?
To rename a module or plugin in a collection, or to redirect a module or plugin to another collection, you need to add a
redirect
entry to
plugin_routing
in
meta/runtime.yml
. For example, to redirect the module
old_cloud
to
foo.bar.new_cloud
, add:
plugin_routing
:
modules
:
old_cloud
:
redirect
:
foo.bar.new_cloud
If you want to deprecate the old name, add a
deprecation:
entry (see above):
plugin_routing
:
modules
:
old_cloud
:
redirect
:
foo.bar.new_cloud
deprecation
:
removal_version
:
2.0.0
warning_text
:
Use foo.bar.new_cloud instead.
You need to use the Fully Qualified Collection Name (FQCN) of the new module/plugin name, even if it is located in the same collection as the redirect. By using a FQCN from another collection, you redirect the module/plugin to that collection.
If you need to support Ansible 2.9, please note that Ansible 2.9 does not know about
meta/runtime.yml
. With Ansible 2.9 you can still rename plugins and modules inside one collection by using symbolic links. Note that ansible-base 2.10, ansible-core 2.11, and newer will prefer
meta/runtime.yml

entries over symbolic links.
Tombstoning a module or plugin in a collection
?
To remove a deprecated module or plugin from a collection, you need to tombstone it:
Remove the module or plugin file with related files like tests, documentation references, and documentation.
Add a tombstone entry in
meta/runtime.yml
. For example, to tombstone the module
old_cloud
, add:
plugin_routing
:
modules
:
old_cloud
:
tombstone
:
removal_version
:
2.0.0
warning_text
:
Use foo.bar.new_cloud instead.
Instead of
removal_version
, you can also use
removal_date
with an ISO 8601 formatted date. The date should be the date of the next major release.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html

Ansible architecture ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Ansible architecture
Edit on GitHub
Ansible architecture
?
Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.
Being designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.
It uses no agents and no additional custom security infrastructure, so it is easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allows you to describe your automation jobs in a way that approaches plain English.
In this section, we?ll give you a really quick overview of how Ansible works so you can see how the pieces fit together.
Modules
Module utilities
Plugins
Inventory
Playbooks
The Ansible search path
Modules
?
Ansible works by connecting to your nodes and pushing out scripts called ?Ansible modules? to them. Most modules accept parameters that describe the desired state of the system.
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.
You can
write your own modules
, though you should first consider
whether you should
. Typically you?ll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content. You may write specialized modules in any language that can return JSON (Ruby, Python, bash, and so on).
Module utilities
?
When multiple modules use the same code, Ansible stores those functions as module utilities to minimize duplication and maintenance. For example, the code that parses URLs is
lib/ansible/module_utils/url.py
. You can
write your own module utilities

as well. Module utilities may only be written in Python or in PowerShell.
Plugins
?
Plugins
augment Ansible?s core functionality. While modules execute on the target system in separate processes (usually that means on a remote system), plugins execute on the control node within the
/usr/bin/ansible
process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more. Ansible ships with a number of handy plugins, and you can easily
write your own
. For example, you can write an
inventory plugin
to connect to any datasource that returns JSON. Plugins must be written in Python.
Inventory
?
By default, Ansible represents the machines it manages in a file (INI, YAML, and so on) that puts all of your managed machines in groups of your own choosing.
To add new machines, there is no additional SSL signing server involved, so there?s never any hassle deciding why a particular machine didn?t get linked up due to obscure NTP or DNS issues.
If there?s another source of truth in your infrastructure, Ansible can also connect to that. Ansible can draw inventory, group, and variable information from sources like EC2, Rackspace, OpenStack, and more.
Here?s what a plain text inventory file looks like:
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called ?group_vars/? or ?host_vars/?) or directly in the inventory file.
Or, as already mentioned, use a dynamic inventory to pull your inventory from data sources like EC2, Rackspace, or OpenStack.
Playbooks
?
Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time. This is where Ansible starts to get most interesting.
Ansible?s approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.
Here?s what a simple playbook looks like:

-
hosts
:
webservers
serial
:
5
update 5 machines at a time
roles
:
-
common
-

webapp
-
hosts
:
content_servers
roles
:
-
common
-
content
The Ansible search path
?
Modules, module utilities, plugins, playbooks, and roles can live in multiple locations. If you write your own code to extend Ansible?s core features, you may have multiple files with similar or the same names in different locations on your Ansible control node. The search path determines which of these files Ansible will discover and use on any given playbook run.
Ansible?s search path grows incrementally over a run. As Ansible finds each playbook and role included in a given run, it appends any directories related to that playbook or role to the search path. Those directories remain in scope for the duration of the run, even after the playbook or role has finished executing. Ansible loads modules, module utilities, and plugins in this order:
Directories adjacent to a playbook specified on the command line. If you run Ansible with
ansible-playbook
/path/to/play.yml
, Ansible appends these directories if they exist:
/path/to/modules
/path/to/module_utils
/path/to/plugins
Directories adjacent to a playbook that is statically imported by a playbook specified on the command line. If
play.yml
includes
-
import_playbook:
/path/to/subdir/play1.yml
, Ansible appends these directories if they exist:
/path/to/subdir/modules
/path/to/subdir/module_utils
/path/to/subdir/plugins
Subdirectories of a role directory referenced by a playbook. If
play.yml
runs
myrole
, Ansible appends these directories if they exist:
/path/to/roles/myrole/modules
/path/to/roles/myrole/module_utils
/path/to/roles/myrole/plugins
Directories specified as default paths in
ansible.cfg
or by the related environment variables, including the paths for the various plugin types. See Ansible Configuration Settings for more information.

Sample
ansible.cfg
fields:
DEFAULT_MODULE_PATH
DEFAULT_MODULE_UTILS_PATH
DEFAULT_CACHE_PLUGIN_PATH
DEFAULT_FILTER_PLUGIN_PATH
Sample environment variables:
ANSIBLE_LIBRARY
ANSIBLE_MODULE_UTILS
ANSIBLE_CACHE_PLUGINS
ANSIBLE_FILTER_PLUGINS
The standard directories that ship as part of the Ansible distribution.
Caution
Modules, module utilities, and plugins in user-specified directories will
override the standard versions. This includes some files with generic names.
For example, if you have a file named
basic.py
in a user-specified
directory, it will override the standard
ansible.module_utils.basic
.
If you have more than one module, module utility, or plugin with the same name in different user-specified directories, the order of commands at the command line and the order of includes and roles in each play will affect which one is found and used on that particular play.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/testing.html

Testing Ansible ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Testing Ansible
Edit on GitHub
Testing Ansible
?
Why test your Ansible contributions?
Types of tests
Testing within GitHub & Azure Pipelines
Organization
Rerunning a failing CI job
How to test a PR
Setup: Installing Pytest and required Pytest libraries
Setup: Checking out a Pull Request
Testing the Pull Request
Run sanity tests
Run unit tests
Run integration tests
Code Coverage Online
Want to know more about testing?
Why test your Ansible contributions?
?
If you?re a developer, one of the most valuable things you can do is to look at GitHub issues and help fix bugs, since bug-fixing is almost always prioritized over feature development. Even for non-developers, helping to test pull requests for bug fixes and features is still immensely valuable.
Ansible users who understand how to write playbooks and roles should be able to test their work. GitHub pull requests will automatically run a variety of tests (for example, Azure Pipelines) that show bugs in action. However, contributors must also test their work outside of the automated GitHub checks and show evidence of these tests in the pull request to ensure that their work will be more likely to be reviewed and merged.
Read on to learn how Ansible is tested, how to test your contributions locally, and how to extend testing capabilities.
If you want to learn about testing collections, read
Testing collections
Types of tests
?
At a high level, we have the following classifications of tests:
sanity
:
Sanity Tests
Sanity tests are made up of scripts and tools used to perform static code analysis.
The primary purpose of these tests is to enforce Ansible coding standards and requirements.

integration
:
Integration tests
Functional tests of modules and Ansible Core functionality.
units
:
Unit Tests
Tests directly against individual parts of the code base.
Testing within GitHub & Azure Pipelines
?
Organization
?
When Pull Requests (PRs) are created they are tested using Azure Pipelines, a Continuous Integration (CI) tool. Results are shown at the end of every PR.
When Azure Pipelines detects an error and it can be linked back to a file that has been modified in the PR then the relevant lines will be added as a GitHub comment. For example:
The test `ansible-test sanity --test pep8` failed with the following errors:

lib/ansible/modules/network/foo/bar.py:509:17: E265 block comment should start with '# '

The test `ansible-test sanity --test validate-modules` failed with the following error:
lib/ansible/modules/network/foo/bar.py:0:0: E307 version_added should be 2.4. Currently 2.3
From the above example we can see that
--test
pep8
and
--test
validate-modules
have identified an issue. The commands given allow you to run the same tests locally to ensure you?ve fixed all issues without having to push your changes to GitHub and wait for Azure Pipelines, for example:
If you haven?t already got Ansible available, use the local checkout by running:
source hacking/env-setup
Then run the tests detailed in the GitHub comment:
ansible-test sanity --test pep8
ansible-test sanity --test validate-modules
If there isn?t a GitHub comment stating what?s failed you can inspect the results by clicking on the ?Details? button under the ?checks have failed? message at the end of the PR.
Rerunning a failing CI job
?
Occasionally you may find your PR fails due to a reason unrelated to your change. This could happen for several reasons, including:
a temporary issue accessing an external resource, such as a yum or Git repo
a timeout creating a virtual machine to run the tests on
If either issue appears to be the case, you can rerun the Azure Pipelines test by:
adding a comment with
/rebuild
(full rebuild) or
/rebuild_failed
(rebuild only failed CI nodes) to the pull request
closing and re-opening the pull request (full rebuild)
making another change to the branch and pushing to GitHub
If the issue persists, please contact the community. Visit the
Ansible communication guide
for details.

How to test a PR
?
Ideally, the code should add tests that prove that the code works. That?s not always possible and tests are not always comprehensive, especially when a user doesn?t have access to a wide variety of platforms, or is using an API or web service. In these cases, live testing against real equipment can be more valuable than automation that runs against simulated interfaces. In any case, things should always be tested manually the first time as well.
Thankfully, helping to test Ansible is pretty straightforward, assuming you are familiar with how Ansible works.
Setup: Installing Pytest and required Pytest libraries
?
Ansible?s unit testing framework leverages the pytest library. Before diving into testing, ensure you have
pytest
installed alongside any additional pytest libraries such as
pytest-mock
and
pytest-xdist
.
Refer to the documentation for more information:
Unit Tests
.
Setup: Checking out a Pull Request
?
You can do this by:
checking out Ansible
fetching the proposed changes into a test branch
testing
commenting on that particular issue on GitHub
Here?s how:
Warning
Testing source code from GitHub pull requests sent to us does have some inherent risk, as the source code sent may have mistakes or malicious code that could have a negative impact on your system. We recommend doing all testing on a virtual machine, whether a cloud instance, or locally. Some users like Vagrant or Docker for this, but they are optional. It is also useful to have virtual machines of different Linux or other flavors, since some features (for example, package managers such as apt or yum) are specific to those OS versions.
Create a fresh area to work:
git clone https://github.com/ansible/ansible.git ansible-pr-testing
cd ansible-pr-testing
Next, find the pull request you?d like to test and make a note of its number. It will look something like this:
Use os.path.sep instead of hardcoding / #65381
Note
Only test
ansible:devel
It is important that the PR request target be
ansible:devel
, as we do not accept pull requests into any other branch. Dot releases are cherry-picked manually by Ansible staff.
Use the pull request number when you fetch the proposed changes and create your branch for testing:
git fetch origin refs/pull/XXXX/head:testing_PRXXXX
git checkout testing_PRXXXX
The first command fetches the proposed changes from the pull request and creates a new branch named
testing_PRXXXX
, where the XXXX is the actual number associated with the pull request (for example, 65381). The second command checks out the newly created branch.
Note
If the GitHub user interface shows that the pull request will not merge cleanly, we do not recommend proceeding if you

are not somewhat familiar with Git and coding, as you will have to resolve a merge conflict. This is the responsibility of the original pull request contributor.
Note
Some users do not create feature branches, which can cause problems when they have multiple, unrelated commits in their version of
devel
. If the source looks like
someuser:devel
, make sure there is only one commit listed on the pull request.
The Ansible source includes a script that allows you to use Ansible directly from source without requiring a
full installation that is frequently used by developers on Ansible.
Simply source it (to use the Linux/Unix terminology) to begin using it immediately:
source ./hacking/env-setup
This script modifies the
PYTHONPATH
environment variables (along with a few other things), which will be temporarily
set as long as your shell session is open.
Testing the Pull Request
?
At this point, you should be ready to begin testing!
Some ideas of what to test are:
Create a test Playbook with the examples in and check if they function correctly
Test to see if any Python backtraces returned (that?s a bug)
Test on different operating systems, or against different library versions
Run sanity tests
?
ansible-test
sanity
More information:
Sanity Tests
Run unit tests
?
ansible-test
units
More information:
Unit Tests
Run integration tests
?
ansible-test
integration
-v
ping
More information:
Integration tests
Any potential issues should be added as comments on the pull request (and it is acceptable to comment if the feature works as well), remembering to include the output of
ansible
--version
Example:
Works for me! Tested on `Ansible 2.3.0`. I verified this on CentOS 6.5 and also Ubuntu 14.04.
If the PR does not resolve the issue, or if you see any failures from the unit/integration tests, just include that output instead:
This change causes errors for me.
When I ran this Ubuntu 16.04 it failed with the following:

```
some output
StackTrace
some other output
```

Code Coverage Online

?

The online code coverage reports
is a good way
to identify areas for testing improvement in Ansible. By following red colors you can
drill down through the reports to find files that have no tests at all. Adding both
integration and unit tests that show clearly how code should work, verify important
Ansible functions and increases testing coverage in areas where there is none is a valuable
way to help improve Ansible.

The code coverage reports only cover the
devel
branch of Ansible where new feature
development takes place. Pull requests and new code will be missing from the codecov.io
coverage reports so local reporting is needed. Most
ansible-test
commands allow you
to collect code coverage, this is particularly useful to indicate where to extend
testing. See
Testing Ansible and Collections
for more information.

Want to know more about testing?

?

If you?d like to know more about the plans for improving testing Ansible then why not join the
Ansible community forum
.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/testing_integration.html

Integration tests ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Integration tests
Edit on GitHub
Integration tests
?
Topics
Integration tests
Quick Start
Configuration
ansible-test command
integration_config.yml
Prerequisites
Non-destructive Tests
Destructive Tests
Windows Tests
Tests in containers
Running Integration Tests
Container Images
Other configuration for Cloud Tests
IAM policies for AWS
testing-policies
Other Definitions required
Network Tests
Where to find out more
The Ansible integration Test system.
Tests for playbooks, by playbooks.
Some tests may require credentials. Credentials may be specified with
credentials.yml
.
Some tests may require root.
Note
Every new module and plugin should have integration tests, even if the tests cannot be run on Ansible CI infrastructure.
In this case, the tests should be marked with the
unsupported
alias in
aliases file
.
Quick Start
?
It is highly recommended that you install and activate the

argcomplete
Python package.
It provides tab completion in
bash
for the
ansible-test
test runner.
Configuration
?
ansible-test command
?
The example below assumes
bin/
is in your
$PATH
. An easy way to achieve that
is to initialize your environment with the
env-setup
command:
source hacking/env-setup
ansible-test --help
You can also call
ansible-test
with the full path:
bin/ansible-test --help
integration_config.yml
?
Making your own version of
integration_config.yml
can allow for setting some
tunable parameters to help run the tests better in your environment. Some
tests (for example, cloud tests) will only run when access credentials are provided. For more
information about supported credentials, refer to the various
cloud-config-*.template
files in the
tests/integration/
directory.
Prerequisites
?
Some tests assume things like
hg
,
svn
, and
git
are installed, and in path. Some tests
(such as those for Amazon Web Services) need separate definitions, which will be covered
later in this document.
(Complete list pending)
Non-destructive Tests
?
These tests will modify files in subdirectories, but will not do things that install or remove packages or things
outside of those test subdirectories. They will also not reconfigure or bounce system services.
Note

Running integration tests within containers
To protect your system from any potential changes caused by integration tests, and to ensure a sensible set of dependencies are available we recommend that you always run integration tests with the
--docker
option, for example
--docker
ubuntu2204
. Get the list of supported container images by running
ansible-test
integration
--help
. You can find them in the
target docker images
section of the output. The
default
image is used for sanity and unit tests, as well as for platform independent integration tests such as those for cloud modules.
Run as follows for all POSIX platform tests executed by our CI system in a Fedora 34 container:
ansible-test integration shippable/ --docker fedora34
You can exclude a specific test as well, such as for individual modules:
ansible-test integration --exclude git
You can target a specific test as well, such as for individual modules:
ansible-test integration ping
You can use the
-v
option to make the output more verbose:
ansible-test integration lineinfile -vvv
Use the following command to list all the available targets:
ansible-test integration --list-targets
Note
Bash users
If you use
bash
with
argcomplete
, obtain a full list by doing:
ansible-test
integration
<tab><tab>
Destructive Tests
?
These tests are allowed to install and remove some trivial packages. You will likely want to devote these to a virtual environment, such as Docker. They won?t reformat your filesystem:
ansible-test integration destructive/ --docker fedora34
Windows Tests
?
These tests exercise the
winrm
connection plugin and Windows modules. You?ll
need to define an inventory with a remote Windows Server to use for testing,
and enable PowerShell Remoting to continue.
Running these tests may result in changes to your Windows host, so don?t run
them against a production/critical Windows environment.
Enable PowerShell Remoting (run on the Windows host by a Remote Desktop):

Enable-PSRemoting -Force
Define Windows inventory:
cp inventory.winrm.template inventory.winrm
$
{
EDITOR:-vi
}
inventory.winrm
Run the Windows tests executed by our CI system:
ansible-test windows-integration -v shippable/
Tests in containers
?
If you have a Linux system with Docker or Podman installed, running integration tests using the same containers used by
the Ansible continuous integration (CI) system is recommended.
Note
Podman
By default, Podman will only be used if the Docker CLI is not installed. If you have Docker installed but want to use Podman, you can change this behavior by setting the environment variable
ANSIBLE_TEST_PREFER_PODMAN
.
Note
Docker on non-Linux
Using Docker Engine to run Docker on a non-Linux host (such as macOS) is not recommended.
Some tests may fail, depending on the image used for testing.
Using the
--docker-privileged
option when running
integration
(not
network-integration
or
windows-integration
) may resolve the issue.
Running Integration Tests
?
To run all CI integration test targets for POSIX platforms in a Ubuntu 18.04 container:
ansible-test integration shippable/ --docker ubuntu1804
You can also run specific tests or select a different Linux distribution.
For example, to run tests for the
ping
module on a Ubuntu 18.04 container:
ansible-test integration ping --docker ubuntu1804
Container Images
?
Container images are updated regularly. To see the current list of container images:
ansible-test
integration
--help
The list is under the
target docker images and supported python version
heading.
Other configuration for Cloud Tests
?

To run some tests, you must provide access credentials in a file named
cloud-config-aws.yml
or
cloud-config-cs.ini
in the tests/integration
directory. Corresponding
.template
files are available for syntax help. The newer AWS
tests now use the file
tests/integration/cloud-config-aws.yml
.

IAM policies for AWS
?

Ansible needs fairly wide ranging powers to run the tests in an AWS account. These rights can be provided to a dedicated user. These need to be configured before running the test.

testing-policies
?

The GitHub repository
mattclay/aws-terminator
contains two sets of policies used for all existing AWS module integration tests.
The
hacking/aws_config/setup_iam.yml
playbook can be used to setup two groups:
ansible-integration-ci
will have the policies applied necessary to run any
integration tests not marked as
unsupported
and are designed to mirror those
used by Ansible?s CI.
ansible-integration-unsupported
will have the additional policies applied
necessary to run the integration tests marked as
unsupported
including tests
for managing IAM roles, users and groups.

Once the groups have been created, you?ll need to create a user and make the user a member of these groups. The policies are designed to minimize the rights of that user. Please note that while this policy does limit the user to one region, this does not fully restrict the user (primarily due to the limitations of the Amazon ARN notation). The user will still have wide privileges for viewing account definitions, and will also be able to manage some resources that are not related to testing (for example, AWS lambdas with different names). Tests should not be run in a primary production account in any case.

Other Definitions required
?

Apart from installing the policy and giving it to the user identity running the tests, a
lambda role
ansible_integration_tests
has to be created which has lambda basic execution
privileges.

Network Tests
?

For guidance on writing network test see
Resource module integration tests
.

Where to find out more

?
If you?d like to know more about the plans for improving testing Ansible, join the Ansible community forum

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/testing_sanity.html

Sanity Tests ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Sanity Tests
?
Topics

Sanity tests are made up of scripts and tools used to perform static code analysis.
The primary purpose of these tests is to enforce Ansible coding standards and requirements.
Tests are run with
ansible-test
sanity
.
All available tests are run unless the
--test
option is used.
Set up your environment
?
Install ansible-core
that provides the
ansible-test
tool.
If you want to run checks available in the development version of
ansible-core
,
install it from source code
.
Run
source
hacking/env-setup
from its source code directory in the same terminal session you run your tests.
Install
podman
or
docker
to avoid installing all the dependencies on your system.
If you test files in a collection:

Ensure you have your collection installed in the following path in your home directory:
~/ansible_collections/<NAMESPACE>/<COLLECTION_NAME>
. For instance, in case of the
community.general
collection, it will be
~/ansible_collections/community/general
If your collection is hosted on a remote server such as GitHub, clone it to that path as follows:
git
clone
<COLLECTION_REPO_URL>
~/ansible_collections/<NAMESPACE>/<COLLECTION_NAME>
How to run
?
Note
To run sanity tests using podman or docker, always use the default docker image
by passing the
--docker
argument without specifying the image name.
When testing files in a collection, change your location to your collection directory you created while
setting up your environment
:
cd
~/ansible_collections/<NAMESPACE>/<COLLECTION_NAME>
To run all sanity tests in a container:
ansible-test
sanity
--docker
To run a
specific test
, add the
--test
<NAME>
argument, for example,
--test
validate-modules
.
To list available tests, run:
ansible-test
sanity
--list-tests
To include disabled tests, add the
--allow-disabled
argument.
Available Tests
?
See the full list of
sanity tests
, which also details how to fix identified issues.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/dev_guide/testing_units.html

Unit Tests ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Unit Tests

?

Unit tests are small isolated tests that target a specific library or module. Unit tests in Ansible are currently the only way of driving tests from python within Ansible?s continuous integration process. This means that in some circumstances the tests may be a bit wider than just units.

Topics

Available Tests

?

Unit tests can be found in
test/units
. Notice that the directory structure of the tests matches that of
lib/ansible/
.

Running Tests

?

Note

To run unit tests using docker, always use the default docker image by passing the
--docker
or
--docker
default
argument.

The Ansible unit tests can be run across the whole code base by doing:

cd
/path/to/ansible/source

source
hacking/env-setup
ansible-test
units
--docker
-v
Against a single file by doing:
ansible-test
units
--docker
-v
apt
Or against a specific Python version by doing:
ansible-test
units
--docker
-v
--python
2
.7
apt
If you are running unit tests against things other than modules, such as module utilities, specify the whole file path:
ansible-test
units
--docker
-v
test/units/module_utils/basic/test_imports.py
For advanced usage see the online help:
ansible-test
units
--help
You can also run tests in Ansible?s continuous integration system by opening a pull
request. This will automatically determine which tests to run based on the changes made
in your pull request.
Installing dependencies
?
If you are running
ansible-test
with the
--docker
or
--venv
option you do not need to install dependencies manually.
Otherwise you can install dependencies using the
--requirements
option, which will
install all the required dependencies needed for unit tests. For example:
ansible-test
units
--python
2
.7
--requirements
apache2_module

The list of unit test requirements can be found at
test/units/requirements.txt
.
This does not include the list of unit test requirements for
ansible-test
itself,
which can be found at
test/lib/ansible_test/_data/requirements/units.txt
.
See also the
constraints
applicable to all test commands.
Extending unit tests
?
Warning
What a unit test isn?t
If you start writing a test that requires external services then
you may be writing an integration test, rather than a unit test.
Structuring Unit Tests
?
Ansible drives unit tests through
pytest
. This
means that tests can either be written a simple functions which are included in any file
name like
test_<something>.py
or as classes.
Here is an example of a function:
#this function will be called simply because it is called test_*()
def
test_add
():
a
=
10
b
=
23
c
=
33
assert
a
+
b
==
c
Here is an example of a class:
import
unittest
class
AddTester
(
unittest

.
TestCase
):
def
SetUp
():
self
.
a
=
10
self
.
b
=
23
this function will
def
test_add
():
c
=
33
assert
self
.
a
+
self
.
b
==
c
this function will
def
test_subtract
():
c
=
-
13
assert
self
.
a
-
self
.
b
==
c

Both methods work fine in most circumstances; the function-based interface is simpler and quicker and so that?s probably where you should start when you are just trying to add a few basic tests for a module. The class-based test allows more tidy set up and tear down

of pre-requisites, so if you have many test cases for your module you may want to refactor to use that.
Assertions using the simple
assert
function inside the tests will give full
information on the cause of the failure with a trace-back of functions called during the assertion. This means that plain asserts are recommended over other external assertion libraries.
A number of the unit test suites include functions that are shared between several modules, especially in the networking arena. In these cases a file is created in the same directory, which is then included directly.
Module test case common code
?
Keep common code as specific as possible within the
test/units/
directory structure.
Don?t import common unit test code from directories outside the current or parent directories.
Don?t import other unit tests from a unit test. Any common code should be in dedicated files that aren?t themselves tests.
Fixtures files
?
To mock out fetching results from devices, or provide other complex data structures that come from external libraries, you can use
fixtures
to read in pre-generated data.
You can check how
fixtures
are used in
CPU_INFO_TEST_SCENARIOS
If you are simulating APIs you may find that Python placebo is useful. See
Unit Testing Ansible Modules
for more information.
Code Coverage For New or Updated Unit Tests
?
New code will be missing from the codecov.io coverage reports (see
Testing Ansible
), so
local reporting is needed. Most
ansible-test
commands allow you to collect code
coverage; this is particularly useful when to indicate where to extend testing.
To collect coverage data add the
--coverage
argument to your
ansible-test
command line:
ansible-test
units
--coverage
apt
ansible-test
coverage
html
Results will be written to

test/results/reports/coverage/index.html
Reports can be generated in several different formats:
ansible-test
coverage
report
- Console report.
ansible-test
coverage
html
- HTML report.
ansible-test
coverage
xml
- XML report.
To clear data between test runs, use the
ansible-test
coverage
erase
command. See
Testing Ansible and Collections
for more information about generating coverage
reports.
See also
Unit Testing Ansible Modules
Special considerations for unit testing modules
Testing Ansible and Collections
Running tests locally including gathering and reporting coverage data
Python 3 documentation - 26.4. unittest ? Unit testing framework
The documentation of the unittest framework in python 3
Python 2 documentation - 25.3. unittest ? Unit testing framework
The documentation of the earliest supported unittest framework - from Python 2.6
pytest: helps you write better programs
The documentation of pytest - the framework actually used to run Ansible unit tests

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/galaxy/user_guide.html

Galaxy User Guide ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Finding collections on Galaxy
Finding roles on Galaxy
Get more information about a role
Installing roles from Galaxy
Installing roles
Setting where to install roles
Installing a specific version of a role
Installing multiple roles from a file
Installing roles and collections from the same requirements.yml file
Installing multiple roles from multiple files

Dependencies
Using
meta/requirements.yml
Using
meta/main.yml
List installed roles
Remove an installed role
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Galaxy User Guide
Edit on GitHub
Galaxy User Guide
?
Ansible Galaxy
refers to the
Galaxy
website, a free site for finding, downloading, and sharing community developed collections and roles.
Use Galaxy to jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work such as
roles
, and
collections
.
The collection format provides a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins. See the
Galaxy documentation
for full details on Galaxy.
Finding collections on Galaxy
Finding roles on Galaxy
Get more information about a role
Installing roles from Galaxy

Installing roles
Installing a specific version of a role
Installing multiple roles from a file
Installing roles and collections from the same requirements.yml file
Installing multiple roles from multiple files
Dependencies
List installed roles
Remove an installed role
Finding collections on Galaxy
?
To find collections on Galaxy:
Click
Collections > Collections
in the left-hand navigation.
Type in your search term. You can filter by keyword, tags, and namespaces.
Galaxy presents a list of collections that match your search criteria.
See
Using Ansible collections
for complete details on installing and using collections.
Finding roles on Galaxy
?
To find standalone roles (that is roles that are not part of a collection):
Click
Roles > Roles
in the left-hand navigation.
Type in your search term. You can filter by keyword, tags, and namespaces.
Galaxy presents a list of roles that match your search criteria.
You can optionally search the Galaxy database by tags, platforms, author and multiple keywords using the
ansible-galaxy
CLI command.
$
ansible-galaxy
role
search
elasticsearch
--author
geerlingguy
The search command will return a list of the first 1000 results matching your search:
Found 6 roles matching your search:

```
 Name                        Description
 ----                        -----------
geerlingguy.elasticsearch        Elasticsearch for Linux.
geerlingguy.elasticsearch-curator Elasticsearch curator for Linux.
geerlingguy.filebeat             Filebeat for Linux.
geerlingguy.fluentd              Fluentd for Linux.
geerlingguy.kibana               Kibana for Linux.
```

Get more information about a role
?
Use the
info
command to view more detail about a specific role:
$
ansible-galaxy

role
info
username.role_name
This returns everything found in Galaxy for the role:
Role: username.role_name
    description: Installs and configures a thing, a distributed, highly available NoSQL thing.
    active: True
    commit: c01947b7bc89ebc0b8a2e298b87ab416aed9dd57
    commit_message: Adding travis
    commit_url: https://github.com/username/repo_name/commit/c01947b7bc89ebc0b8a2e298b87ab
    company: My Company, Inc.
    created: 2015-12-08T14:17:52.773Z
    download_count: 1
    forks_count: 0
    github_branch: main
    github_repo: repo_name
    github_user: username
    id: 6381
    is_valid: True
    issue_tracker_url:
    license: Apache
    min_ansible_version: 2.15
    modified: YYYY-MM-DDTHH:MM:SS.000Z
    namespace: username
    open_issues_count: 0
    path: /Users/username/projects/roles
    role_type: ANS
    stargazers_count: 0
    travis_status_url: https://travis-ci.org/username/repo_name.svg?branch=main
Installing roles from Galaxy
?
The
ansible-galaxy
command comes bundled with Ansible, and you can use it to install roles from Galaxy or directly from a Git based SCM. You can
also use it to create a new role, remove roles, or perform tasks on the Galaxy website.
The command line tool by default communicates with the Galaxy website API using the server address
https://galaxy.ansible.com
. If you run your own internal Galaxy server
and want to use it instead of the default one, pass the
--server
option followed by the address of this galaxy server. You can set this option permanently by setting
the Galaxy server value in your
ansible.cfg
file. See
GALAXY_SERVER
for details on setting the value in
ansible.cfg
.
Installing roles
?
Use the
ansible-galaxy
command to download roles from the

Galaxy website
$
ansible-galaxy
role
install
namespace.role_name
Setting where to install roles
?
By default, Ansible downloads roles to the first writable directory in the default list of paths
~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles
. This installs roles in the home directory of the user running
ansible-galaxy
.
You can override this with one of the following options:
Set the environment variable
ANSIBLE_ROLES_PATH
in your session.
Use the
--roles-path
option for the
ansible-galaxy
command.
Define
roles_path
in an
ansible.cfg
file.
The following provides an example of using
--roles-path
to install the role into the current working directory:
$
ansible-galaxy
role
install
--roles-path
.
geerlingguy.apache
See also
Configuring Ansible
All about configuration files
Installing a specific version of a role
?
When the Galaxy server imports a role, it imports any Git tags matching the
Semantic Version
format as versions.
In turn, you can download a specific version of a role by specifying one of the imported tags.
To see the available versions for a role:
Locate the role on the Galaxy search page.
Click on the name to view more details, including the available versions.
To install a specific version of a role from Galaxy, append a comma and the value of a GitHub release tag. For example:
$
ansible-galaxy
role
install

geerlingguy.apache,3.2.0
It is also possible to point directly to the Git repository and specify a branch name or commit hash as the version. For example, the following will
install a specific commit:
$
ansible-galaxy
role
install
git+https://github.com/geerlingguy/ansible-role-apache.git,0b7cd353c0250e87a26e0499e59e7fd265cc2f25
Installing multiple roles from a file
?
You can install multiple roles by including the roles in a
requirements.yml
file. The format of the file is YAML, and the
file extension must be either
.yml
or
.yaml
.
Use the following command to install roles included in
requirements.yml:
$
ansible-galaxy
install
-r
requirements.yml
Again, the extension is important. If the
.yml
extension is left off, the
ansible-galaxy
CLI assumes the file is in an older, now deprecated,
?basic? format.
Each role in the file will have one or more of the following attributes:
src
The source of the role. Use the format
namespace.role_name
, if downloading from Galaxy; otherwise, provide a URL pointing
to a repository within a Git based SCM. See the examples below. This is a required attribute.
scm
Specify the SCM. As of this writing only
git
or
hg
are allowed. See the examples below. Defaults to
git
.
version:
The version of the role to download. Provide a release tag value, commit hash, or branch name. Defaults to the branch set as a default in the repository, otherwise defaults to the
master
.
name:
Download the role to a specific name. Defaults to the Galaxy name when downloading from Galaxy, otherwise it defaults to the name of the repository.

Use the following example as a guide for specifying roles in requirements.yml:

```
# from galaxy
- name: yatesr.timezone
# from locally cloned Git repository (git+file:// requires full paths)
- src: git+file:///home/bennojoy/nginx
# from GitHub
- src: https://github.com/bennojoy/nginx
# from GitHub, overriding the name and specifying a specific tag
- name: nginx_role
  src: https://github.com/bennojoy/nginx
  version: main
# from GitHub, specifying a specific commit hash
- src: https://github.com/bennojoy/nginx
  version: "ee8aa41"
# from a webserver, where the role is packaged in a tar.gz
- name: http-role-gz
  src: https://some.webserver.example.com/files/main.tar.gz
# from a webserver, where the role is packaged in a tar.bz2
- name: http-role-bz2
  src: https://some.webserver.example.com/files/main.tar.bz2
# from a webserver, where the role is packaged in a tar.xz (Python 3.x only)
```

-
name
:
http-role-xz
src
:
https://some.webserver.example.com/files/main.tar.xz
from Bitbucket
-
src
:
git+https://bitbucket.org/willthames/git-ansible-galaxy
version
:
v1.4
from Bitbucket, alternative syntax and caveats
-
src
:
https://bitbucket.org/willthames/hg-ansible-galaxy
scm
:
hg
from GitLab or other git-based scm, using git+ssh
-
src
:

:mygroup/ansible-core.git
scm
:
git
version
:
"0.1"
quoted, so YAML doesn't parse this as a floating-point value
Warning
Embedding credentials into a SCM URL is not secure. Make sure to use safe auth options for security reasons. For example, use
SSH
,
netrc
or
http.extraHeader
/
url.<base>.pushInsteadOf
in Git config to prevent your credentials from being exposed in logs.
Installing roles and collections from the same requirements.yml file
?
You can install roles and collections from the same requirements files
---
roles
:
Install a role from Ansible Galaxy.

-
name
:
geerlingguy.java
version
:
"1.9.6"
note that ranges are not supported for roles
collections
:
Install a collection from Ansible Galaxy.
-
name
:
community.general
version
:
">=7.0.0"
source
:
https://galaxy.ansible.com
Installing multiple roles from multiple files
?
For large projects, the
include
directive in a
requirements.yml
file provides the ability to split a large file into multiple smaller files.
For example, a project may have a
requirements.yml
file, and a
webserver.yml
file.
Below are the contents of the
webserver.yml
file:
from github
-
src:
https://github.com/bennojoy/nginx
from Bitbucket
-
src:
git+https://bitbucket.org/willthames/git-ansible-galaxy
version:
v1.4
The following shows the contents of the
requirements.yml
file that now includes the
webserver.yml
file:
from galaxy
-
name:

yatesr.timezone
-
include:
<path_to_requirements>/webserver.yml
To install all the roles from both files, pass the root file, in this case
requirements.yml
on the
command line, as follows:
$
ansible-galaxy
role
install
-r
requirements.yml
Dependencies
?
Roles can also be dependent on other roles, and when you install a role that has dependencies, those dependencies will automatically be installed to the
roles_path
.
There are two ways to define the dependencies of a role:
using
meta/requirements.yml
using
meta/main.yml
Using
meta/requirements.yml
?
New in version 2.10.
You can create the file
meta/requirements.yml
and define dependencies in the same format used for
requirements.yml
described in the
Installing multiple roles from a file
section.
From there, you can import or include the specified roles in your tasks.
Using
meta/main.yml
?
Alternatively, you can specify role dependencies in the
meta/main.yml
file by providing a list of roles under the
dependencies
section. If the source of a role is Galaxy, you can simply specify the role in the format
namespace.role_name
. You can also use the more complex format in
requirements.yml
, allowing you to provide
src
,
scm
,

version
, and
name
.
Dependencies installed that way, depending on other factors described below, will also be executed
before
this role is executed during play execution.
To better understand how dependencies are handled during play execution, see
Roles
.
The following shows an example
meta/main.yml
file with dependent roles:
---
dependencies
:
-
geerlingguy.java
galaxy_info
:
author
:
geerlingguy
description
:
Elasticsearch for Linux.
company
:
"Midwestern
Mac,
LLC"
license
:
"license
(BSD,
MIT)"
min_ansible_version
:
2.4
galaxy_tags
:
-
web
-
system
-
monitoring
-
logging
-
lucene
-
elk
-

elasticsearch
Tags are inherited
down
the dependency chain. In order for tags to be applied to a role and all its dependencies, the tag should be applied to the role, not to all the tasks within a role.
Roles listed as dependencies are subject to conditionals and tag filtering, and may not execute fully depending on what tags and conditionals are applied.
If the source of a role is Galaxy, specify the role in the format
namespace.role_name
:
dependencies
:
-
geerlingguy.apache
-
geerlingguy.ansible
Alternately, you can specify the role dependencies in the complex form used in
requirements.yml
as follows:
dependencies
:
-
name
:
geerlingguy.ansible
-
name
:
composer
src
:
git+https://github.com/geerlingguy/ansible-role-composer.git
version
:
775396299f2da1f519f0d8885022ca2d6ee80ee8
Note
Galaxy expects all role dependencies to exist in Galaxy, and therefore dependencies to be specified in the
namespace.role_name
format. If you import a role with a dependency where the
src
value is a URL, the import process will fail.
List installed roles
?
Use
list
to show the name and version of each role installed in the
roles_path
.
$
ansible-galaxy
role
list
-
namespace-1.foo,

v2.7.2
-
namespace2.bar,
v2.6.2
Remove an installed role
?
Use
remove
to delete a role from
roles_path
:
$
ansible-galaxy
role
remove
namespace.role_name
See also
Using Ansible collections
Shareable collections of modules, playbooks and roles
Roles
Reusable tasks, handlers, and other files in a known directory structure
Working with command line tools
Perform other related operations
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/getting_started/index.html

Getting started with Ansible ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Introduction to Ansible
Start automating with Ansible
Building an inventory
Creating a playbook
Ansible concepts
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords

Getting started with Ansible
?
Ansible automates the management of remote systems and controls their desired state.
As shown in the preceding figure, most Ansible environments have three main components:
Control node
A system on which Ansible is installed.
You run Ansible commands such as
ansible
or
ansible-inventory
on a control node.
Inventory
A list of managed nodes that are logically organized.
You create an inventory on the control node to describe host deployments to Ansible.
Managed node
A remote system, or host, that Ansible controls.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/index.html

Ansible Documentation ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Ansible Documentation
Ansible Documentation
?
Welcome to Ansible community documentation!
This documentation covers the version of Ansible noted in the upper left corner of this page.
We maintain multiple versions of Ansible and of the documentation, so please be sure you are using the version of the documentation that covers the version of Ansible you?re using.
For recent features, we note the version of Ansible where the feature was added.
Ansible releases a new major release approximately twice a year.
The core application evolves somewhat conservatively, valuing simplicity in language design and setup.
Contributors develop and change modules and plugins, hosted in collections, much more quickly.
Ansible getting started
Getting started with Ansible
Introduction to Ansible
Start automating with Ansible
Building an inventory
Creating a playbook
Ansible concepts
Getting started with Execution Environments
Introduction to Execution Environments
Setting up your environment
Building your first Execution Environment
Running your EE
Running Ansible with the community EE image
Installation, Upgrade & Configuration
Installation Guide
Installing Ansible
Installing Ansible on specific operating systems
Configuring Ansible
Ansible Porting Guides
Ansible 12 Porting Guide
Ansible 11 Porting Guide
Ansible 10 Porting Guide
Ansible 9 Porting Guide
Ansible 8 Porting Guide
Ansible 7 Porting Guide
Ansible 6 Porting Guide
Ansible 5 Porting Guide
Ansible 4 Porting Guide

YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
Ansible project 12.0
Ansible project 11.0
Ansible project 10.0
Ansible project 9.0
Ansible project 8.0
Ansible project 7.0
Ansible project 6.0
Ansible project 5.0
Ansible project 4.0
Ansible project 3.0
Ansible project 2.10
Older Roadmaps
ansible-core Roadmaps
Ansible-core 2.19
Ansible-core 2.18
Ansible-core 2.17
Ansible-core 2.16
Ansible-core 2.15
Ansible-core 2.14
Ansible-core 2.13
Ansible-core 2.12
Ansible-core 2.11
Ansible-base 2.10
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Installing Ansible ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Installing Ansible
Control node requirements
Managed node requirements
Node requirement summary
Selecting an Ansible package and version to install
Installing and upgrading Ansible with pipx
Installing Ansible
Upgrading Ansible
Installing Extra Python Dependencies
Installing and upgrading Ansible with pip
Locating Python
Ensuring
pip
is available
Installing Ansible
Upgrading Ansible
Installing Ansible to containers
Installing for development
Installing
devel
from GitHub with
pip
Running the
devel
branch from a clone
Confirming your installation
Adding Ansible command shell completion
Installing
argcomplete
Configuring
argcomplete
Installing Ansible on specific operating systems
Configuring Ansible
Ansible Porting Guides
Using Ansible
Building Ansible inventories

Ansible
Installation Guide
Installing Ansible
Edit on GitHub
Installing Ansible
?
Ansible is an agentless automation tool that you install on a single host (referred to as the control node).

From the control node, Ansible can manage an entire fleet of machines and other devices (referred to as managed nodes) remotely with SSH, Powershell remoting, and numerous other transports, all from a simple command-line interface with no databases or daemons required.
Control node requirements
Managed node requirements
Node requirement summary
Selecting an Ansible package and version to install
Installing and upgrading Ansible with pipx
Installing Ansible
Upgrading Ansible
Installing Extra Python Dependencies
Installing and upgrading Ansible with pip
Locating Python
Ensuring
pip
is available
Installing Ansible
Upgrading Ansible
Installing Ansible to containers
Installing for development
Installing
devel
from GitHub with
pip
Running the
devel
branch from a clone
Confirming your installation
Adding Ansible command shell completion
Installing
argcomplete
Configuring
argcomplete
Global configuration
Per command configuration
Using
argcomplete
with zsh or tcsh
Control node requirements
?
For your
control
node (the machine that runs Ansible), you can use nearly any UNIX-like machine with Python installed. This includes Red Hat, Debian, Ubuntu, macOS, BSDs, and Windows under a
Windows Subsystem for Linux (WSL) distribution
. Windows without WSL is not natively supported as a control node; see
Matt Davis? blog post
for more information.
Managed node requirements
?
The
managed
node (the machine that Ansible is managing) does not require Ansible to be installed, but requires Python to run Ansible-generated Python code.

The managed node also needs a user account that can connect through SSH to the node with an interactive POSIX shell.

Note

There can be exceptions in module requirements. For example, network modules do not require Python on the managed device. See documentation for the modules you use.

Node requirement summary

?

You can find details about control and managed node requirements, including Python versions, for each Ansible version in the

ansible-core control node Python support

and

ansible-core support matrix

sections.

Selecting an Ansible package and version to install

?

Ansible?s community packages are distributed in two ways:

ansible-core

: a minimalist language and runtime package containing a set of

built-in modules and plugins

.

ansible

: a much larger ?batteries included? package, which adds a community-curated selection of

Ansible Collections

for automating a wide variety of devices.

Choose the package that fits your needs.

The following instructions use

ansible

as a package name, but you can substitute

ansible-core

if you prefer to start with the minimal package and separately install only the Ansible Collections you require.

The

ansible

or

ansible-core

packages may be available in your operating systems package manager, and you are free to install these packages with your preferred method. For more information, see the

Installing Ansible on specific operating systems

guide. These installation instructions only cover the officially supported means of installing the python packages with

pip

.

See the

Ansible package release status table

for the

ansible-core

version included in the package.

Installing and upgrading Ansible with pipx

?

On some systems, it may not be possible to install Ansible with

pip

, due to decisions made by the operating system developers. In such cases,

pipx

is a widely available alternative.

These instructions will not go over the steps to install

pipx

; if those instructions are needed, please continue to the
pipx installation instructions
for more information.
Installing Ansible
?
Use
pipx
in your environment to install the full Ansible package:
$
pipx
install
--include-deps
ansible
You can install the minimal
ansible-core
package:
$
pipx
install
ansible-core
Alternately, you can install a specific version of
ansible-core
:
$
pipx
install
ansible-core
==
2
.12.3
Upgrading Ansible
?
To upgrade an existing Ansible installation to the latest released version:
$
pipx
upgrade
--include-injected
ansible
Installing Extra Python Dependencies
?
To install additional python dependencies that may be needed, with the example of installing the
argcomplete
python package as described below:
$
pipx
inject
ansible
argcomplete
Include the
--include-apps
option to make apps in the additional python dependency available on your PATH. This allows you to execute commands for those apps from the shell.
$
pipx

inject
--include-apps
ansible
argcomplete
If you need to install dependencies from a requirements file, for example when installing the Azure collection, you can use
runpip
.
$
pipx
runpip
ansible
install
-r
~/.ansible/collections/ansible_collections/azure/azcollection/requirements.txt
Installing and upgrading Ansible with pip
?
Locating Python
?
Locate and remember the path to the Python interpreter you wish to use to run Ansible. The following instructions refer to this Python as
python3
.
For example, if you have determined that you want the Python at
/usr/bin/python3.9
to be the one that you will install Ansible under, specify that instead of
python3
.
Ensuring
pip
is available
?
To verify whether
pip
is already installed for your preferred Python:
$
python3
-m
pip
-V
If all is well, you should see something like the following:
$
python3
-m
pip
-V
pip 21.0.1 from /usr/lib/python3.9/site-packages/pip (python 3.9)
If so,
pip
is available, and you can move on to the
next step
.
If you see an error like
No

module
named
pip
, you will need to install
pip
under your chosen Python interpreter before proceeding.
This may mean installing an additional OS package (for example,
python3-pip
), or installing the latest
pip
directly from the Python Packaging Authority by running the following:
$
curl
https://bootstrap.pypa.io/get-pip.py
-o
get-pip.py
$
python3
get-pip.py
--user
You may need to perform some additional configuration before you are able to run Ansible. See the Python documentation on
installing to the user site
for more information.
Installing Ansible
?
Use
pip
in your selected Python environment to install the full Ansible package for the current user:
$
python3
-m
pip
install
--user
ansible
You can install the minimal
ansible-core
package for the current user:
$
python3
-m
pip
install
--user
ansible-core
Alternately, you can install a specific version of
ansible-core
:
$
python3
-m
pip
install

--user
ansible-core
==
2
.12.3
Upgrading Ansible
?
To upgrade an existing Ansible installation in this Python environment to the latest released version, simply add
--upgrade
to the command above:
$
python3
-m
pip
install
--upgrade
--user
ansible
Installing Ansible to containers
?
Instead of installing Ansible content manually, you can simply build an execution environment container image or use one of the available community images as your control node.
See
Getting started with Execution Environments
for details.
Installing for development
?
If you are testing new features, fixing bugs, or otherwise working with the development team on changes to the core code, you can install and run the source from GitHub.
Note
You should only install and run the
devel
branch if you are modifying
ansible-core
or trying out features under development. This is a rapidly changing source of code and can become unstable at any point.
For more information on getting involved in the Ansible project, see the
Ansible Community Guide
.
For more information on creating Ansible modules and Collections, see the
Developer Guide
.
Installing
devel
from GitHub with
pip
?
You can install the
devel
branch of
ansible-core
directly from GitHub with
pip
:

$
python3
-m
pip
install
--user
https://github.com/ansible/ansible/archive/devel.tar.gz
You can replace
devel
in the URL mentioned above, with any other branch or tag on GitHub to install older versions of Ansible, tagged alpha or beta versions, and release candidates.
Running the
devel
branch from a clone
?
ansible-core
is easy to run from source. You do not need
root
permissions to use it and there is no software to actually install. No daemons or database setup are required.
Clone the
ansible-core
repository
$
git
clone
https://github.com/ansible/ansible.git
$
cd
./ansible
Setup the Ansible environment
Using Bash
$
source
./hacking/env-setup
Using Fish
$
source
./hacking/env-setup.fish
To suppress spurious warnings/errors, use
-q
$
source
./hacking/env-setup
-q
Install Python dependencies
$
python3
-m
pip
install
--user
-r
./requirements.txt
Update the

devel
branch of
ansible-core
on your local machine
Use pull-with-rebase so any local changes are replayed.
$
git
pull
--rebase
Confirming your installation
?
You can test that Ansible is installed correctly by checking the version:
$
ansible
--version
The version displayed by this command is for the associated
ansible-core
package that has been installed.
To check the version of the
ansible
package that has been installed:
$
ansible-community
--version
Adding Ansible command shell completion
?
You can add shell completion of the Ansible command line utilities by installing an optional dependency called
argcomplete
. It supports bash, and has limited support for zsh and tcsh.
For more information about installation and configuration, see the
argcomplete documentation
.
Installing
argcomplete
?
If you chose the
pipx
installation instructions:
$
pipx
inject
--include-apps
ansible
argcomplete
If you chose the
pip
installation instructions:
$
python3
-m
pip
install
--user
argcomplete

Configuring
argcomplete
?
There are 2 ways to configure
argcomplete
to allow shell completion of the Ansible command line utilities: globally or per command.
Global configuration
?
Global completion requires bash 4.2.
$
activate-global-python-argcomplete
--user
This will write a bash completion file to a user location. Use
--dest
to change the location or
sudo
to set up the completion globally.
Per command configuration
?
If you do not have bash 4.2, you must register each script independently.
$
eval
$(
register-python-argcomplete
ansible
)
$
eval
$(
register-python-argcomplete
ansible-config
)
$
eval
$(
register-python-argcomplete
ansible-console
)
$
eval
$(
register-python-argcomplete
ansible-doc
)
$
eval
$(
register-python-argcomplete
ansible-galaxy
)
$
eval
$(
register-python-argcomplete

ansible-inventory
)
$
eval
$(
register-python-argcomplete
ansible-playbook
)
$
eval
$(
register-python-argcomplete
ansible-pull
)
$
eval
$(
register-python-argcomplete
ansible-vault
)
You should place the above commands into your shell?s profile file such as
~/.profile
or
~/.bash_profile
.
Using
argcomplete
with zsh or tcsh
?
See the
argcomplete documentation
.
See also
Introduction to ad hoc commands
Examples of basic commands
Working with playbooks
Learning ansible?s configuration management language
How do I handle the package dependencies required by Ansible package dependencies during Ansible installation ?
Ansible Installation related to FAQs
Forum
Join the Ansible community forum to get help and share insights
Real-time chat
How to join Ansible chat channels
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/inventory_guide/intro_dynamic_inventory.htm

Working with dynamic inventory ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
How to build your inventory
Working with dynamic inventory
Inventory script example: Cobbler
Other inventory scripts
Using inventory directories and multiple inventory sources
Static groups of dynamic groups
Patterns: targeting hosts and groups
Connection methods and details
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices

Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Building Ansible inventories
Working with dynamic inventory
Edit on GitHub
Working with dynamic inventory
?
Inventory script example: Cobbler
Other inventory scripts
Using inventory directories and multiple inventory sources
Static groups of dynamic groups
If your Ansible inventory fluctuates over time, with hosts spinning up and shutting down in response to business demands, the static inventory solutions described in
How to build your inventory
will not serve your needs.
You may need to track hosts from multiple sources: cloud providers, LDAP,
Cobbler
, and/or enterprise CMDB systems.
Ansible integrates all of these options through a dynamic external inventory system.
Ansible supports two ways to connect with external inventory:
Inventory plugins
and
inventory scripts
.
Inventory plugins take advantage of the most recent updates to the Ansible Core code.
We recommend plugins over scripts for dynamic inventory.
You can
write your own plugin
to connect to additional dynamic inventory sources.
You can still use inventory scripts if you choose.
When we implemented inventory plugins, we ensured backwards compatibility through the script inventory plugin.
The examples below illustrate how to use inventory scripts.
If you prefer a GUI for handling dynamic inventory, the inventory database on AWX or

syncs with all your dynamic inventory sources, provides web and REST access to the results, and offers a graphical inventory editor.
With a database record of all of your hosts, you can correlate past event history and see which hosts have had failures on their last playbook runs.
Inventory script example: Cobbler
?
Ansible integrates seamlessly with
Cobbler
, a Linux installation server originally written by Michael DeHaan and now led by James Cammarata, who works for Ansible.
While primarily used to kickoff OS installations and manage DHCP and DNS, Cobbler has a generic
layer that can represent data for multiple configuration management systems (even at the same time) and serve as a ?lightweight CMDB?.
To tie your Ansible inventory to Cobbler, copy
this script
to
/etc/ansible
and
chmod
+x
the file. Run
cobblerd
any time you use Ansible and use the
-i
command line option (for example,
-i
/etc/ansible/cobbler.py
) to communicate with Cobbler using Cobbler?s XMLRPC API.
Add a
cobbler.ini
file in
/etc/ansible
so Ansible knows where the Cobbler server is and some cache improvements can be used. For example:

```
[cobbler]

# Set Cobbler's hostname or IP address
host = http://127.0.0.1/cobbler_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API
# call. Set this to the path you want cache files to be written to. Two files
# will be written to this directory:
#   - ansible-cobbler.cache
#   - ansible-cobbler.index

cache_path = /tmp

# The number of seconds a cache file is considered valid. After this many
# seconds, a new API call will be made, and the cache file will be updated.

cache_max_age = 900
```

First test the script by running
/etc/ansible/cobbler.py
directly. You should see some JSON data output, but it may not have anything in it just yet.

Let?s explore what this does. In Cobbler, assume a scenario somewhat like the following:

```
cobbler
profile
add
--name
=
webserver
--distro
=
CentOS6-x86_64
cobbler
profile
edit
--name
=
webserver
--mgmt-classes
=
"webserver"
--ksmeta
=
"a=2 b=3"
cobbler
system
edit
--name
=
foo
--dns-name
=
"foo.example.com"
--mgmt-classes
=
"atlanta"
--ksmeta
=
"c=4"
cobbler
system
edit
--name
=
bar
--dns-name
=
"bar.example.com"
--mgmt-classes
=
"atlanta"
--ksmeta
=
"c=5"
```

In the example above, the system ?foo.example.com? is addressable by ansible directly, but is also addressable when using the group names ?webserver? or ?atlanta?. Since Ansible uses SSH, it contacts system foo over

?foo.example.com?, only, never just ?foo?. Similarly, if you tried ?ansible foo?, it would not find the system? but ?ansible ?foo*?? would do, because the system DNS name starts with ?foo?.
The script provides more than host and group info. In addition, as a bonus, when the ?setup? module is run (which happens automatically when using playbooks), the variables ?a?, ?b?, and ?c? will all be auto-populated in the templates:
file: /srv/motd.j2
Welcome, I am templated with a value of a={{ a }}, b={{ b }}, and c={{ c }}
Which could be executed just like this:
ansible
webserver
-m
setup
ansible
webserver
-m
template
-a
"src=/tmp/motd.j2 dest=/etc/motd"
Note
The name ?webserver? came from Cobbler, as did the variables for
the config file. You can still pass in your own variables like
normal in Ansible, but variables from the external inventory script
will override any that have the same name.
So, with the template above (
motd.j2
), this results in the following data being written to
/etc/motd
for system ?foo?:
Welcome, I am templated with a value of a=2, b=3, and c=4
And on system ?bar? (bar.example.com):
Welcome, I am templated with a value of a=2, b=3, and c=5
And technically, though there is no major good reason to do it, this also works:
ansible
webserver
-m
ansible.builtin.shell
-a
"echo {{ a }}"
So, in other words, you can use those variables in arguments/actions as well.
Other inventory scripts
?
In Ansible 2.10 and later, inventory scripts moved to their associated collections. Many are now in the
ansible-community/contrib-scripts repository
. We recommend you use
Inventory plugins
instead.
Using inventory directories and multiple inventory sources
?
If the location given to
-i
in Ansible is a directory (or as so configured in
ansible.cfg
), Ansible can use multiple inventory sources
at the same time. When doing so, it is possible to mix both dynamic and statically managed inventory sources in the

same ansible run. Instant
hybrid cloud!
In an inventory directory, executable files are treated as dynamic inventory sources and most other files as static sources. Files which end with any of the following are ignored:
~, .orig, .bak, .ini, .cfg, .retry, .pyc, .pyo
You can replace this list with your own selection by configuring an
inventory_ignore_extensions
list in
ansible.cfg
, or setting the
ANSIBLE_INVENTORY_IGNORE
environment variable. The value in either case must be a comma-separated list of patterns, as shown above.
Any
group_vars
and
host_vars
subdirectories in an inventory directory are interpreted as expected, making inventory directories a powerful way to organize different sets of configurations. See
Passing multiple inventory sources
for more information.
Static groups of dynamic groups
?
When defining groups of groups in the static inventory file, the child groups
must also be defined in the static inventory file, otherwise ansible returns an
error. If you want to define a static group of dynamic child groups, define
the dynamic groups as empty in the static inventory file. For example:
[tag_Name_staging_foo]

[tag_Name_staging_bar]

[staging:children]
tag_Name_staging_foo
tag_Name_staging_bar
See also
How to build your inventory
All about static inventory files
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

How to build your inventory ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
How to build your inventory
Inventory basics: formats, hosts, and groups
Default groups
Hosts in multiple groups
Grouping groups: parent/child group relationships
Adding ranges of hosts
Passing multiple inventory sources
Organizing inventory in a directory
Managing inventory load order
Adding variables to inventory
Assigning a variable to one machine: host variables
Inventory aliases
Defining variables in INI format
Assigning a variable to many machines: group variables
Inheriting variable values: group variables for groups of groups
Organizing host and group variables
How variables are merged
Managing inventory variable load order
Connecting to hosts: behavioral inventory parameters
Non-SSH connection types
Inventory setup examples
Example: One inventory per environment
Example: Group by function
Example: Group by location
Working with dynamic inventory
Patterns: targeting hosts and groups
Connection methods and details
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX

How to build your inventory

?

Ansible automates tasks on managed nodes or ?hosts? in your infrastructure by using a list or group of lists known as inventory. Ansible composes its inventory from one or more ?inventory sources?. While one of these sources can be the list of host names you pass at the command line, most Ansible users create inventory files. Your inventory defines the managed nodes you automate and the variables associated with those hosts. You can also specify groups. Groups allow you to reference multiple associated hosts to target for your automation or to define variables in bulk.

Once you define your inventory, you use

patterns

to select the hosts or groups you want Ansible to run against.
The simplest inventory is a single file that contains a list of hosts and groups. The default location for this file is
/etc/ansible/hosts
. You can specify a different inventory source or sources at the command line by using the
-i
<path
or
expression>
option or by using the configuration system.
Ansible
Inventory plugins
supports a range of formats and sources, which makes your inventory flexible and customizable. As your inventory expands, you might need more than a single file to organize your hosts and groups. You have the following common options beyond the
/etc/ansible/hosts
file:
You can generate an inventory dynamically. For example, you can use an inventory plugin to list resources in one or more cloud providers or other sources. See
Working with dynamic inventory
.
You can use multiple sources for inventory, including both dynamic inventory and static files. See
Passing multiple inventory sources
.
You can create a directory with multiple inventory sources, static or dynamic. See
Organizing inventory in a directory
.
Inventory basics: formats, hosts, and groups
Default groups
Hosts in multiple groups
Grouping groups: parent/child group relationships
Adding ranges of hosts
Passing multiple inventory sources
Organizing inventory in a directory
Managing inventory load order
Adding variables to inventory
Assigning a variable to one machine: host variables
Inventory aliases
Defining variables in INI format
Assigning a variable to many machines: group variables
Inheriting variable values: group variables for groups of groups
Organizing host and group variables
How variables are merged
Managing inventory variable load order
Connecting to hosts: behavioral inventory parameters
Non-SSH connection types
Inventory setup examples
Example: One inventory per environment
Example: Group by function
Example: Group by location
The following YAML snippets include an ellipsis (?) to indicate that the snippets are part of a larger YAML file. You can find out more about YAML syntax at
YAML Basics
.
Inventory basics: formats, hosts, and groups

?
You can create your inventory file in one of many formats, depending on the inventory plugins you have.
The most common formats are INI and YAML because Ansible includes built-in support for them. This introduction focuses on these two formats, but many other formats and sources are possible.
A basic INI
/etc/ansible/hosts
might look like this:
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
The headings in brackets are group names. You can use group names to classify hosts
and to decide which hosts you are controlling at what times and for what purpose.
Group names should follow the same guidelines as
Creating valid variable names
.
Here?s the same basic inventory file in YAML format:
ungrouped
:
hosts
:
mail.example.com
:
webservers
:
hosts
:
foo.example.com
:
bar.example.com
:
dbservers
:
hosts
:
one.example.com
:
two.example.com
:
three.example.com
:
Default groups
?
Even if you do not define any groups in your inventory, Ansible creates two default groups:
all
and
ungrouped
. The

all
group contains every host. The
ungrouped
group contains all hosts that do not belong to any other group.
Every host always belongs to at least two groups (
all
and
ungrouped
, or
all
and another group). For example, in the basic inventory above, the host
mail.example.com
belongs to the
all
and
ungrouped
groups. The host
two.example.com
belongs to the
all
and
dbservers
groups. Although
all
and
ungrouped
are always present, they can be implicit and might not appear in group listings like
group_names
.
Hosts in multiple groups
?
You can put a host in more than one group. For example, you can include a production web server in a data center in Atlanta in the
[prod]
,
[atlanta]
, and
[webservers]
groups. You can create groups that track the following criteria:
What
- An application, stack, or microservice (for example, database servers, web servers, and so on).
Where
- A datacenter or region, to talk to local DNS, storage, and so on (for example, east, west).
When
- The development stage, to avoid testing on production resources (for example, prod, test).
The following example extends the previous YAML inventory to include what, when, and where:
ungrouped
:
hosts
:
mail.example.com
:
webservers
:

hosts
:
foo.example.com
:
bar.example.com
:
dbservers
:
hosts
:
one.example.com
:
two.example.com
:
three.example.com
:
east
:
hosts
:
foo.example.com
:
one.example.com
:
two.example.com
:
west
:
hosts
:
bar.example.com
:
three.example.com
:
prod
:
hosts
:
foo.example.com
:
one.example.com
:
two.example.com
:
test
:
hosts
:
bar.example.com
:
three.example.com
:
As the example shows,
one.example.com

exists in the
dbservers
,
east
, and
prod
groups.
Grouping groups: parent/child group relationships
?
You can create parent/child relationships among groups. Parent groups are also known as nested groups or groups of groups. For example, if all your production hosts are already in groups such as
atlanta_prod
and
denver_prod
, you can create a
production
group that includes those smaller groups. This approach reduces maintenance because you add or remove hosts from the parent group by editing the child groups.
To create parent/child relationships for groups, use one of the following methods:
In INI format, use the
:children
suffix.
In YAML format, use the
children:
entry.
The following example shows the same inventory as above, simplified with parent groups for the
prod
and
test
groups:
ungrouped
:
hosts
:
mail.example.com
:
webservers
:
hosts
:
foo.example.com
:
bar.example.com
:
dbservers
:
hosts
:
one.example.com
:
two.example.com
:
three.example.com
:

east
:
hosts
:
foo.example.com
:
one.example.com
:
two.example.com
:
west
:
hosts
:
bar.example.com
:
three.example.com
:
prod
:
children
:
east
:
test
:
children
:
west
:
Note the following properties of child groups:
Any host that is a member of a child group is automatically a member of the parent group.
A group can have multiple parents and children, but not circular relationships.
A host can be in multiple groups, but Ansible processes only
one
instance of the host at runtime. Ansible merges the data from multiple groups.
Hosts and groups are always ?global?. If you define a host or group more than once under different ?branches? or ?instances?, the host or group remains the same entity. Defining a host or group more than once either adds new information to it or overwrites any conflicting information with the latest definition.
Adding ranges of hosts
?
Some plugins, like YAML and INI, support adding ranges of hosts. If you have many hosts with a similar pattern, you can add the hosts as a range rather than listing each hostname separately:
In INI:
[webservers]
www[01:50].example.com
In YAML:
...
webservers
:
hosts
:
www[01:50].example.com
:

You can specify a stride (increments between sequence numbers) when you define a numeric range of hosts:
In INI:
[webservers]
www[01:50:2].example.com
In YAML:
...
webservers
:
hosts
:
www[01:50:2].example.com
:
The example above matches the subdomains www01, www03, www05, ?, www49, but not www00, www02, www50, and so on, because the stride (increment) is 2 units for each step.
For numeric patterns, you can include or remove leading zeros as desired. Ranges are inclusive. You can also define alphabetic ranges:
[databases]
db-[a:f].example.com
Passing multiple inventory sources
?
You can target multiple inventory sources (static files, directories, dynamic inventory scripts
or anything supported by inventory plugins) at the same time. To do this, specify multiple inventory sources from the command
line (see below) or by configuration, either by setting
ANSIBLE_INVENTORY
or in
ansible.cfg
(
DEFAULT_HOST_LIST
).
This capability can be useful when you want to target normally separate environments, like staging and production, at the same time for a specific action.
To target two inventory sources from the command line:
ansible-playbook
get_logs.yml
-i
staging
-i
production
Organizing inventory in a directory
?
You can consolidate multiple inventory sources in a single directory. The simplest version of this approach is a directory with multiple files instead of a single inventory file. Maintaining a single file becomes difficult when the file gets too long. If you have multiple teams and multiple automation projects, creating one inventory file per team or project lets everyone easily find the hosts and groups that matter to them. You can also still use the files individually or in subsets, depending on how you configure or call Ansible.
These files can use all formats or plugin configurations (for example, YAML or INI). In this case, your directory becomes your ?single? inventory source, and Ansible aggregates the multiple sources it finds in that directory. By default, Ansible ignores some directories and extensions, but you can change this behavior in the configuration (
INVENTORY_IGNORE_PATTERNS
and
INVENTORY_IGNORE_EXTS
).
You can also combine multiple inventory source types in an inventory directory. This method can be useful for

combining static and dynamic hosts and managing them as one inventory.
The following inventory directory combines an inventory plugin source, a dynamic inventory script, and a file with static hosts:
inventory/
  openstack.yml   # configure inventory plugin to get hosts from OpenStack cloud
  dynamic-inventory.py   # add additional hosts with dynamic inventory script
  on-prem   # add static hosts and groups
  parent-groups   # add static hosts and groups
You can target this inventory directory as follows:
ansible-playbook
example.yml
-i
inventory
You can also configure the inventory directory in your
ansible.cfg
file. See
Configuring Ansible
for more details.
Ansible reads and loads files from the top directory down in alphabetically sorted order.
Managing inventory load order
?
Ansible loads inventory sources in the order you supply them. It defines hosts, groups, and variables as it encounters them in the source files, adding the
all
and
ungrouped
groups at the end if needed.
Depending on the inventory plugin or plugins you use, you might need to rearrange the order of sources to ensure that parent/child-defined groups or hosts exist as the plugins expect. Otherwise, you might encounter a parsing error. For example, the YAML and INI inventory plugins discard empty groups (groups with no associated hosts) when they finish processing each source.
If you define a variable multiple times, Ansible overwrites the previous value. The last definition wins.
Adding variables to inventory
?
You can define variables that relate to a specific host or group in your inventory. A simple way to start is by adding variables directly to the hosts and groups in a YAML or INI inventory source.
This guide documents how to add variables in the inventory source for simplicity. However, you can also use
Vars plugins
to add variables from many other sources. By default, Ansible ships with the
host_group_vars
plugin, which allows you to define variables in separate host and group variable files. Using separate files is a more robust approach to describing your system policy than defining variables in the inventory source. See
Organizing host and group variables
for guidelines on how to store variable values in individual files in the ?host_vars? and ?group_vars? directories.
Assigning a variable to one machine: host variables
?
You can easily assign a variable to a single host and then use that variable later in playbooks. You can do this directly in your inventory file.
In INI:
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
In YAML:
atlanta

:
hosts
:
host1
:
http_port
:
80
maxRequestsPerChild
:
808
host2
:
http_port
:
303
maxRequestsPerChild
:
909
Unique values like non-standard SSH ports work well as host variables. You can add them to your Ansible inventory by adding the port number after the hostname with a colon:
badwolf.example.com:5309
You can use host variables to define ?Connection variables?. Connection variables configure
connection
,
shell
, and
become
plugins to enable task execution on the host. For example:
[targets]

localhost ansible_connection=local
other1.example.com ansible_connection=ssh ansible_user=myuser
other2.example.com ansible_connection=ssh ansible_user=myotheruser
Inventory aliases
?
The
inventory_hostname
is the unique identifier for a host in Ansible. This identifier can be an IP address or a hostname, but it can also be just an ?alias? or short name for the host.
In INI:
jumper ansible_port=5555 ansible_host=192.0.2.50
In YAML:
...
hosts
:
jumper
:
ansible_port
:
5555
ansible_host
:
192.0.2.50

In this example, running Ansible against the host alias ?jumper? connects to 192.0.2.50 on port 5555. See
behavioral inventory parameters
to further customize the connection to hosts.
This feature is also useful for targeting the same host more than once, but remember that tasks can run in parallel:
In INI:
jumper1 ansible_port=5555 ansible_host=192.0.2.50
jumper2 ansible_port=5555 ansible_host=192.0.2.50
In YAML:
...
hosts
:
jumper1
:
ansible_port
:
5555
ansible_host
:
192.0.2.50
jumper2
:
ansible_port
:
5555
ansible_host
:
192.0.2.50
Defining variables in INI format
?
Ansible interprets values that you pass in the INI format by using the
key=value
syntax differently depending on where you declare them:
When you declare a value inline with the host, Ansible interprets the INI value as a Python literal structure (for example, a string, number, tuple, list, dict, boolean, or None). Host lines accept multiple
key=value
parameters per line. Therefore, you need a way to indicate that a space is part of a value rather than a separator. You can quote values that contain whitespace (using single or double quotes). See the
Python shlex parsing rules
for details.
When you declare a value in a
:vars
section, Ansible interprets the INI value as a string. For example,
var=FALSE
creates a string with the value ?FALSE?. Unlike host lines,
:vars
sections accept only a single entry per line, so everything after the
=
becomes the value for the entry.
If you need a variable from an INI inventory to have a certain type (for example, a string or a boolean), always specify the type with a filter in your task. Do not rely on types that you set in INI inventories when you consume variables.
Consider using the YAML format for inventory sources to avoid confusion about the actual type of a variable. The YAML inventory plugin processes variable values consistently and correctly.
Assigning a variable to many machines: group variables
?

If all hosts in a group share a variable value, you can apply that variable to an entire group at once.
In INI:
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
In YAML:
atlanta
:
hosts
:
host1
:
host2
:
vars
:
ntp_server
:
ntp.atlanta.example.com
proxy
:
proxy.atlanta.example.com
Group variables are a convenient way to apply variables to multiple hosts at once. Before executing, however, Ansible always flattens variables, including inventory variables, to the host level. If a host is a member of multiple groups, Ansible reads variable values from all of those groups. If you assign different values to the same variable in different groups, Ansible chooses which value to use based on internal
rules for merging
.
Inheriting variable values: group variables for groups of groups
?
You can apply variables to parent groups (nested groups or groups of groups) as well as to child groups. The syntax is the same:
:vars
for INI format and
vars:
for YAML format:
In INI:
[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
```

In YAML:

```
usa
:
children
:
southeast
:
children
:
atlanta
:
hosts
:
host1
:
host2
:
raleigh
:
hosts
:
host2
:
host3
:
vars
:
some_server
:
foo.southeast.example.com
halon_system_timeout
:
30
self_destruct_countdown
:
60
escape_pods
:
2
northeast
:
northwest
:
southwest
```

:
A child group?s variables have higher precedence (they override) than a parent group?s variables.
Organizing host and group variables
?
Although you can define variables in the inventory source, you can also use
Vars plugins
to define alternate sources for your variables.
The default vars plugin that Ansible ships with,
host_group_vars
, lets you use separate host and group variable files. This method helps you organize your variable values more easily. You can also use lists and hash data in these files, which you cannot do in your main inventory file.
For the
host_group_vars
plugin, your host and group variable files must use YAML syntax. Valid file extensions are ?.yml?, ?.yaml?, ?.json?, or no file extension. See
YAML Syntax
if you are new to YAML.
The
host_group_vars
plugin loads host and group variable files by searching paths relative to the inventory source or the playbook file. If your inventory file at
/etc/ansible/hosts
contains a host named ?foosball? that belongs to the
raleigh
and
webservers
groups, that host will use variables from the YAML files in the following locations:
/etc/ansible/group_vars/raleigh
can optionally end in '.yml', '.yaml', or '.json'
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
For example, if you group hosts in your inventory by datacenter, and each datacenter uses its own NTP server and database server, you can create a file named
/etc/ansible/group_vars/raleigh
to store the variables for the
raleigh
group:
---
ntp_server
:
acme.example.org
database_server
:
storage.example.org
You can also create
directories
named after your groups or hosts. Ansible reads all the files in these directories in lexicographical order. Here is an example with the ?raleigh? group:
/etc/ansible/group_vars/raleigh/db_settings
/etc/ansible/group_vars/raleigh/cluster_settings
All hosts in the ?raleigh? group have the variables that you define in these files
available to them. This method is very useful for keeping your variables organized when a single
file gets too big, or when you want to use
Ansible Vault

on some group variables.
Ansible?s
host_group_vars
vars plugin can also add
group_vars/
and
host_vars/
directories to your playbook directory when you use
ansible-playbook
. However, not all Ansible commands have a playbook (for example,
ansible
or
ansible-console
). For those commands, you can use the
--playbook-dir
option to provide the directory on the command line.
If you have sources for the vars plugins relative to both the playbook directory and the inventory directory, the variables that Ansible sources relative to the playbook override the variables that it sources relative to the inventory source.
To track changes to your inventory and variable definitions, keep your inventory sources and their relative variable directories and files in a Git repository or other version control system.
How variables are merged
?
Note
Ansible merges variables from different sources and applies precedence to some variables over others according to a set of rules. For example, variables that occur higher in an inventory can override variables that occur lower in the inventory. See
Variable precedence: where should I put a variable?
for more information.
Before it runs a play, Ansible merges and flattens variables to the specific host. This process keeps Ansible focused on the Host and Task, so groups do not survive outside of inventory and host matching. By default, Ansible overwrites variables, including the ones that you define for a group or host (see
DEFAULT_HASH_BEHAVIOUR
). The order/precedence for inventory entities is (from lowest to highest):
The following list shows the order of precedence for inventory entities, from lowest to highest:
all
group (because it is the ?parent? of all other groups)
parent group
child group
host
By default, Ansible merges groups at the same parent/child level in alphabetical order. Variables from the last group that Ansible loads overwrite variables from the previous groups. For example, Ansible merges an
a_group
with a
b_group
, and matching variables from
b_group
overwrite the variables in
a_group
.
You can fine-tune this merge behavior by setting the group variable
ansible_group_priority
. This variable overrides the alphabetical sorting for the merge order for groups of the same level (after Ansible resolves the parent/child order). The larger the number, the later Ansible merges the group, giving it higher priority. This variable defaults to

1
if you do not set it. For example:
a_group
:
vars
:
testvar
:
a
ansible_group_priority
:
10
b_group
:
vars
:
testvar
:
b
In this example, if both groups have the same priority, the result would normally be
testvar
==
b
. However, because we give
a_group
a higher priority, the result is
testvar
==
a
.
You can set
ansible_group_priority
only in an inventory source, not in
group_vars/
. Ansible uses this variable when it loads the
group_vars/
directory.
Managing inventory variable load order
?
This section describes how to control variable precedence by managing the load order of inventory sources. You can pass sources in a specific order at the command line or use prefixes in the filenames of sources within a directory.
When you use multiple inventory sources, remember that Ansible resolves any variable conflicts according to
the rules described in
How variables are merged
and
Variable precedence: where should I put a variable?
. You can control the merging order of variables in inventory sources to get the variable value you need.
When you pass multiple inventory sources at the command line, Ansible merges variables in the order you pass those parameters. If the
[all:vars]
section in the staging inventory defines
myvar
=
1

and the production inventory defines
myvar
=
2
, then the following outcomes are true:
If you pass
-i
staging
-i
production
, Ansible runs the playbook with
myvar
=
2
.
If you pass
-i
production
-i
staging
, Ansible runs the playbook with
myvar
=
1
.
When you put multiple inventory sources in a directory, Ansible merges the sources in alphabetical order according to their filenames. You can control the load order by adding prefixes to the files:

```
inventory/
  01-openstack.yml          # configure inventory plugin to get hosts from Openstack cloud
  02-dynamic-inventory.py   # add additional hosts with dynamic inventory script
  03-static-inventory       # add static hosts
  group_vars/
    all.yml                 # assign variables to all hosts
```

If
01-openstack.yml
defines
myvar
=
1
for the group
all
,
02-dynamic-inventory.py
defines
myvar
=
2
,
and
03-static-inventory
defines
myvar
=
3

, Ansible runs the playbook with
myvar
=
3
.
For more details on inventory plugins and dynamic inventory scripts see
Inventory plugins
and
Working with dynamic inventory
.
Connecting to hosts: behavioral inventory parameters
?
As described above, you can set the following variables to control how Ansible interacts with remote hosts.
Host connection:
Note
Ansible does not expose a channel to allow communication between the user and the ssh process to accept a password manually to decrypt an ssh key when using the ssh connection plugin (which is the default). The use of
ssh-agent
is highly recommended.
ansible_connection
Specifies the connection type to the host. This can be the name of any Ansible connection plugin. SSH protocol types are
ssh
or
paramiko
. The default is
ssh
.
General for all connections:
ansible_host
Specifies the resolvable name or IP of the host to connect to, if it is different from the alias you wish to give to it. Never set it to depend on
inventory_hostname
. If you really need something like that, use
inventory_hostname_short
so it can work with delegation.
ansible_port
The connection port number, if not the default (22 for ssh).
ansible_user
The username to use when connecting (logging in) to the host.
ansible_password
The password to use to authenticate to the host. (Never store this variable in plain text. Always use a vault. See
Keep vaulted variables safely visible
.)
Specific to the SSH connection plugin:
ansible_ssh_private_key_file
Private key file used by SSH. This is useful if you use multiple keys and you do not want to use SSH agent.
ansible_ssh_common_args
Ansible always appends this setting to the default command line for
sftp
,
scp
,
and

ssh
. This is useful for configuring a
ProxyCommand
for a certain host or
group.
ansible_sftp_extra_args
Ansible always appends this setting to the default
sftp
command line.
ansible_scp_extra_args
Ansible always appends this setting to the default
scp
command line.
ansible_ssh_extra_args
Ansible always appends this setting to the default
ssh
command line.
ansible_ssh_pipelining
Specifies whether to use SSH pipelining. This can override the
pipelining
setting in
ansible.cfg
.
ansible_ssh_executable (added in version 2.2)
This setting overrides the default behavior to use the system
ssh
. It can override the
ssh_executable
setting in the
ssh_connection
section of
ansible.cfg
.
Privilege escalation (see
Ansible Privilege Escalation
for further details):
ansible_become
Equivalent to
ansible_sudo
or
ansible_su
; allows you to force privilege escalation.
ansible_become_method
Allows you to set the privilege escalation method to a matching become plugin.
ansible_become_user
Equivalent to
ansible_sudo_user
or
ansible_su_user
; allows you to set the user you become through privilege escalation.
ansible_become_password
Equivalent to
ansible_sudo_password
or

ansible_su_password
; allows you to set the privilege escalation password. (Never store this variable in plain text. Always use a vault. See Keep vaulted variables safely visible
.)
ansible_become_exe
Equivalent to
ansible_sudo_exe
or
ansible_su_exe
; allows you to set the executable for the escalation method you selected.
ansible_become_flags
Equivalent to
ansible_sudo_flags
or
ansible_su_flags
; allows you to set the flags passed to the selected escalation method. You can also set this globally in
ansible.cfg
in the
become_flags
option under
privilege_escalation
.
Remote host environment parameters:
ansible_shell_type
Specifies the shell type of the target system. You should not use this setting unless you have set the
ansible_shell_executable
to a non-Bourne (sh) compatible shell. By default, Ansible
formats commands using
sh
-style syntax. If you set this to
csh
or
fish
, commands
that Ansible executes on target systems follow those shell?s syntax instead.
ansible_python_interpreter
Specifies the target host Python path. This is useful for systems with more
than one Python or for systems where Python is not located at
/usr/bin/python
, such as *BSD, or where
/usr/bin/python
is not a 2.X series Python. We do not use the
/usr/bin/env
mechanism because that requires the remote user?s
path to be set correctly and also assumes the
python
executable is named python, where the executable might
be named something like
python2.6
.
ansible_*_interpreter
Works for any language, such as Ruby or Perl, and works just like
ansible_python_interpreter
.

This variable replaces the shebang of modules that will run on that host.
New in version 2.1.
ansible_shell_executable
This setting sets the shell the Ansible control node will use on the target machine.
It overrides
executable
in
ansible.cfg
, which defaults to
/bin/sh
. You should only change this value if it is not possible
to use
/bin/sh
(in other words, if
/bin/sh
is not installed on the target
machine or cannot be run from sudo.).
Examples from an Ansible-INI host file:
some_host   ansible_port=2222   ansible_user=manager
aws_host   ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
freebsd_host   ansible_python_interpreter=/usr/local/bin/python
ruby_module_host ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
Non-SSH connection types
?
As stated in the previous section, Ansible executes playbooks over SSH by default, but it is not limited to this connection type.
You can change the connection type with the host-specific parameter
ansible_connection=<connection
plugin
name>
.
For a full list of available plugins and examples, see
Plugin list
.
Inventory setup examples
?
See also
Sample Ansible setup
, which shows inventory along with playbooks and other Ansible artifacts.
Example: One inventory per environment
?
If you need to manage multiple environments, consider defining only the
hosts of a single environment in each inventory. This
way, it is harder to, for example, accidentally change the state of
nodes inside the ?test? environment when you wanted to update
some ?staging? servers.
For the example mentioned above, you could have an
inventory_test
file:
[dbservers]
db01.test.example.com
db02.test.example.com
[appservers]
app01.test.example.com

app02.test.example.com
app03.test.example.com
That file only includes hosts that are part of the ?test?
environment. You can define the ?staging? machines in another file
called
inventory_staging
:
[dbservers]
db01.staging.example.com
db02.staging.example.com
[appservers]
app01.staging.example.com
app02.staging.example.com
app03.staging.example.com
To apply a playbook called
site.yml
to all the app servers in the test environment, use the
following command:
ansible-playbook
-i
inventory_test
-l
appservers
site.yml
Example: Group by function
?
In the previous section, you already saw an example of using groups to
cluster hosts that have the same function. This approach allows you,
for example, to define firewall rules inside a playbook or role
that affect 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
Example: Group by location
?
Other tasks might focus on where a certain host is located. Let?s

say that
db01.test.example.com
and
app01.test.example.com
are
located in DC1, while
db02.test.example.com
is in DC2:
[dc1]
db01.test.example.com
app01.test.example.com
[dc2]
db02.test.example.com
In practice, you might end up mixing all these setups. For example, you
might need to update all nodes in a specific data center
on one day, while on another day, you might need to update all the application servers no matter
their location.
See also
Inventory plugins
Pulling inventory from dynamic or static sources
Working with dynamic inventory
Pulling inventory from dynamic sources, such as cloud providers
Introduction to ad hoc commands
Examples of basic commands
Working with playbooks
Learning Ansible?s configuration, deployment, and orchestration language.
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/module_plugin_guide/modules_intro.html

Introduction to modules ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Introduction to modules
Boolean variables
Module maintenance and support
Rejecting modules
Working with plugins
Modules and plugins index
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins

Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Using Ansible modules and plugins
Introduction to modules
Edit on GitHub
Introduction to modules
?
Modules (also referred to as ?task plugins? or ?library plugins?) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote managed node, and collects return values. In Ansible 2.10 and later, most modules are hosted in collections.
You can execute modules from the command line.
ansible webservers -m service -a "name=httpd state=started"
ansible webservers -m ping
ansible webservers -m command -a "/sbin/reboot -t now"
Each module supports arguments. Nearly all modules take
key=value
arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.
From playbooks, Ansible modules are executed in a very similar way.
-
name
:
reboot the servers
command
:
/sbin/reboot -t now
Another way to pass arguments to a module is using YAML syntax, also called ?complex args?.
-
name
:
restart webserver
service
:
name
:

httpd
state
:
restarted
All modules return JSON format data. This means modules can be written in any programming language. Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state. When used in an Ansible playbook, modules can trigger ?change events? in the form of notifying
handlers
to run additional tasks.
You can access the documentation for each module from the command line with the ansible-doc tool.
ansible-doc yum
For a list of all available modules, see the
Collection docs
, or run the following at a command prompt.
ansible-doc -l
Boolean variables
?
Ansible accepts a broad range of values for
bool
in module arguments:
true/false
,
1/0
,
yes/no
,
True/False
and so on. The matching of valid strings is case insensitive.
While documentation examples focus on
true/false
to be compatible with
ansible-lint
default settings, you can use any of the following:
Valid values
Description
True
,
'true'
,
't'
,
'yes'
,
'y'
,
'on'
,
'1'
,
1
,
1.0
Truthy values
False

,
'false'
,
'f'
,
'no'
,
'n'
,
'off'
,
'0'
,
0
,
0.0
Falsy values
See also
Introduction to ad hoc commands
Examples of using modules in /usr/bin/ansible
Working with playbooks
Examples of using modules with /usr/bin/ansible-playbook
Should you develop a module?
How to write your own modules
Python API
Examples of using modules with the Python API
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Indexes of all modules and plugins
All modules and plugins available
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html

Conditionals ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Ansible playbooks
Working with playbooks
Templating (Jinja2)
Using filters to manipulate data
Tests
Lookups
Python3 in templates
The now function: get the current time
The undef function: add hint for undefined variables
Loops
Controlling where tasks run: delegation and local actions
Conditionals
Basic conditionals with
when
Debugging conditionals
Commonly-used facts
Blocks
Handlers: running operations on change
Error handling in playbooks
Setting the remote environment
Working with language-specific version managers
Reusing Ansible artifacts
Roles
Module defaults
Interactive input: prompts
Using variables
Discovering variables: facts and magic variables
Playbook Example: Continuous Delivery and Rolling Upgrades
Executing playbooks
Advanced playbook syntax
Manipulating data

Conditionals
?
In a playbook, you may want to execute different tasks or have different goals, depending on the value of a fact (data about the remote system), a variable, or the result of a previous task. You may want the value of some variables to

depend on the value of other variables. Or you may want to create additional groups of hosts based on whether the hosts match other criteria. You can do all of these things with conditionals.
Ansible uses Jinja2
tests
and
filters
in conditionals. Ansible supports all the standard tests and filters and adds some unique ones as well.
Note
There are many options to control execution flow in Ansible. You can find more examples of supported conditionals at https://jinja.palletsprojects.com/en/latest/templates/#comparisons
.
Basic conditionals with
when
Conditionals based on ansible_facts
Conditions based on registered variables
Conditionals based on variables
Using conditionals in loops
Loading custom facts
Conditionals with reuse
Conditionals with imports
Conditionals with includes
Conditionals with roles
Selecting variables, files, or templates based on facts
Selecting variables files based on facts
Selecting files and templates based on facts
Debugging conditionals
Commonly-used facts
ansible_facts[?distribution?]
ansible_facts[?distribution_major_version?]
ansible_facts[?os_family?]
Basic conditionals with
when
?
The simplest conditional statement applies to a single task. Create the task, then add a
when
statement that applies a test. The
when
clause is a raw Jinja2 expression without double curly braces (see
Referencing simple variables
). When you run the task or playbook, Ansible evaluates the test for all hosts. On any host where the test passes (returns a value of True), Ansible runs that task. For example, if you are installing mysql on multiple machines, some of which have SELinux enabled, you might have a task to configure SELinux to allow mysql to run. You would only want that task to run on machines that have SELinux enabled:
tasks
:
-
name
:
Configure SELinux to start mysql on any port
ansible.posix.seboolean
:
name
:
mysql_connect_any

state
:
true
persistent
:
true
when
:
ansible_selinux.status == "enabled"
# all variables can be used directly in conditionals without double curly braces
Conditionals based on ansible_facts
?
Often you want to execute or skip a task based on facts. Facts are attributes of individual hosts, including IP address, operating system, the status of a filesystem, and many more. With conditionals based on facts:
You can install a certain package only when the operating system is a particular version.
You can skip configuring a firewall on hosts with internal IP addresses.
You can perform cleanup tasks only when a filesystem is getting full.
See
Commonly-used facts
for a list of facts that frequently appear in conditional statements. Not all facts exist for all hosts. For example, the ?lsb_major_release? fact used in the example below only exists when the
lsb_release
package
is installed on the target host. To see what facts are available on your systems, add a debug task to your playbook:
-
name
:
Show facts available on the system
ansible.builtin.debug
:
var
:
ansible_facts
Here is a sample conditional based on a fact:
tasks
:
-
name
:
Shut down Debian flavored systems
ansible.builtin.command
:
/sbin/shutdown -t now
when
:
ansible_facts['os_family'] == "Debian"
If you have multiple conditions, you can group them with parentheses:
tasks
:
-
name
:
Shut down CentOS 6 and Debian 7 systems
ansible.builtin.command

:
/sbin/shutdown -t now
when
:
(ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_major_version'] == "6") or
(ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "7")
You can use
logical operators
to combine conditions. When you have multiple conditions that all need to be true (that is, a logical
and
), you can specify them as a list:
tasks
:
-
name
:
Shut down CentOS 6 systems
ansible.builtin.command
:
/sbin/shutdown -t now
when
:
-
ansible_facts['distribution'] == "CentOS"
-
ansible_facts['distribution_major_version'] == "6"
If a fact or variable is a string, and you need to run a mathematical comparison on it, use a filter to ensure that Ansible reads the value as an integer:
tasks
:
-
ansible.builtin.shell
:
echo "only on Red Hat 6, derivatives, and later"
when
:
ansible_facts['os_family'] == "RedHat" and ansible_facts['lsb']['major_release'] | int >= 6
You can store Ansible facts as variables to use for conditional logic, as in the following example:
tasks
:
-
name
:
Get the CPU temperature
set_fact
:
temperature
:
"{{
ansible_facts['cpu_temperature']
}}"
-
name
:

Restart the system if the temperature is too high
when
:
temperature | float > 90
shell
:
"reboot"
Conditions based on registered variables
?
Often in a playbook, you want to execute or skip a task based on the outcome of an earlier task. For example, you might want to configure a service after it is upgraded by an earlier task. To create a conditional based on a registered variable:
Register the outcome of the earlier task as a variable.
Create a conditional test based on the registered variable.
You create the name of the registered variable using the
register
keyword. A registered variable always contains the status of the task that created it as well as any output that the task generated. You can use registered variables in templates and action lines as well as in conditional
when
statements. You can access the string contents of the registered variable using
variable.stdout
. For example:
-
name
:
Test play
hosts
:
all
tasks
:
-
name
:
Register a variable
ansible.builtin.shell
:
cat /etc/motd
register
:
motd_contents
-
name
:
Use the variable in conditional statement
ansible.builtin.shell
:
echo "motd contains the word hi"
when
:
motd_contents.stdout.find('hi') != -1
You can use registered results in the loop of a task if the variable is a list. If the variable is not a list, you can convert it into a list, with either
stdout_lines
or with

variable.stdout.split()
. You can also split the lines by other fields:
-
name
:
Registered variable usage as a loop list
hosts
:
all
tasks
:
-
name
:
Retrieve the list of home directories
ansible.builtin.command
:
ls /home
register
:
home_dirs
-
name
:
Add home dirs to the backup spooler
ansible.builtin.file
:
path
:
/mnt/bkspool/{{ item }}
src
:
/home/{{ item }}
state
:
link
loop
:
"{{
home_dirs.stdout_lines
}}"
same as loop: "{{ home_dirs.stdout.split() }}"
The string content of a registered variable can be empty. If you want to run another task only on hosts where the stdout of your registered variable is empty, check the registered variable?s string contents for emptiness:
-
name
:
check registered variable for emptiness
hosts
:
all
tasks
:
-

name
:
List contents of directory
ansible.builtin.command
:
ls mydir
register
:
contents
-
name
:
Check contents for emptiness
ansible.builtin.debug
:
msg
:
"Directory
is
empty"
when
:
contents.stdout == ""
Ansible always registers something in a registered variable for every host, even on hosts where a task fails or Ansible skips a task because a condition is not met. To run a follow-up task on these hosts, query the registered variable for
is
skipped
(not for ?undefined? or ?default?). See
Registering variables
for more information. Here are sample conditionals based on the success or failure of a task. Remember to ignore errors if you want Ansible to continue executing on a host when a failure occurs:
tasks
:
-
name
:
Register a variable, ignore errors and continue
ansible.builtin.command
:
/bin/false
register
:
result
ignore_errors
:
true
-
name
:
Run only if the task that registered the "result" variable fails
ansible.builtin.command
:
/bin/something
when

:
result is failed
-
name
:
Run only if the task that registered the "result" variable succeeds
ansible.builtin.command
:
/bin/something_else
when
:
result is succeeded
-
name
:
Run only if the task that registered the "result" variable is skipped
ansible.builtin.command
:
/bin/still/something_else
when
:
result is skipped
-
name
:
Run only if the task that registered the "result" variable changed something.
ansible.builtin.command
:
/bin/still/something_else
when
:
result is changed
Note
Older versions of Ansible used
success
and
fail
, but
succeeded
and
failed
use the correct tense. All of these options are now valid.
Conditionals based on variables
?
You can also create conditionals based on variables defined in the playbooks or inventory. Because conditionals require boolean input (a test must evaluate as True to trigger the condition), you must apply the
|
bool
filter to non-boolean variables, such as string variables with content like ?yes?, ?on?, ?1?, or ?true?. You can define variables like this:
vars
:
epic
:

true
monumental
:
"yes"
With the variables above, Ansible would run one of these tasks and skip the other:
tasks
:
-
name
:
Run the command if "epic" or "monumental" is true
ansible.builtin.shell
:
echo "This certainly is epic!"
when
:
epic or monumental | bool
-
name
:
Run the command if "epic" is false
ansible.builtin.shell
:
echo "This certainly isn't epic!"
when
:
not epic
If a required variable has not been set, you can skip or fail using Jinja2?s
defined
test. For example:
tasks
:
-
name
:
Run the command if "foo" is defined
ansible.builtin.shell
:
echo "I've got '{{ foo }}' and am not afraid to use it!"
when
:
foo is defined
-
name
:
Fail if "bar" is undefined
ansible.builtin.fail
:
msg="Bailing out. This play requires 'bar'"
when
:
bar is undefined
This is especially useful in combination with the conditional import of
vars

files (see below).
As the examples show, you do not need to use
{{ }}
to use variables inside conditionals, as these are already implied.
Using conditionals in loops
?
If you combine a
when
statement with a
loop
, Ansible processes the condition separately for each item. This is by design, so you can execute the task on some items in the loop and skip it on other items. For example:
tasks
:
-
name
:
Run with items greater than 5
ansible.builtin.command
:
echo {{ item }}
loop
:
[
0
,
2
,
4
,
6
,
8
,
10
]
when
:
item > 5
If you need to skip the whole task when the loop variable is undefined, use the
|default
filter to provide an empty iterator. For example, when looping over a list:
-
name
:
Skip the whole task when a loop variable is undefined
ansible.builtin.command
:
echo {{ item }}
loop
:
"{{
mylist|default([])
}}"

when
:
item > 5
You can do the same thing when looping over a dict:
-
name
:
The same as above using a dict
ansible.builtin.command
:
echo {{ item.key }}
loop
:
"{{
query('dict',
mydict|default({}))
}}"
when
:
item.value > 5
Loading custom facts
?
You can provide your own facts, as described in
Should you develop a module?
. To run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and the variables returned there will be accessible for future tasks:
tasks
:
-
name
:
Gather site specific fact data
action
:
site_facts
-
name
:
Use a custom fact
ansible.builtin.command
:
/usr/bin/thingy
when
:
my_custom_fact_just_retrieved_from_the_remote_system == '1234'
Conditionals with reuse
?
You can use conditionals with reusable tasks files, playbooks, or roles. Ansible executes these conditional statements differently for dynamic reuse (includes) and static reuse (imports). See
Reusing Ansible artifacts
for more information on reuse in Ansible.
Conditionals with imports
?
When you add a conditional to an import statement, Ansible applies the condition to all tasks within the imported file.

This behavior is the equivalent of
Tag inheritance: adding tags to multiple tasks
. Ansible applies the condition to every task and evaluates each task separately. For example, if you want to define and then display a variable that was not previously defined, you might have a playbook called
main.yml
and a tasks file called
other_tasks.yml
:

```
# all tasks within an imported file inherit the condition from the import statement
# main.yml
-
hosts
:
all
tasks
:
-
import_tasks
:
other_tasks.yml
# note "import"
when
:
x is not defined
# other_tasks.yml
-
name
:
Set a variable
ansible.builtin.set_fact
:
x
:
foo
-
name
:
Print a variable
ansible.builtin.debug
:
var
:
x
```

Ansible expands this at execution time to the equivalent of:

```
-
name
:
Set a variable if not defined
ansible.builtin.set_fact
:
x
:
foo
when
```

:
x is not defined
this task sets a value for x
-
name
:
Do the task if "x" is not defined
ansible.builtin.debug
:
var
:
x
when
:
x is not defined
Ansible skips this task, because x is now defined
If
x
is initially defined, both tasks are skipped as intended. But if
x
is initially undefined, the debug task will be skipped since the conditional is evaluated for every imported task. The conditional will evaluate to
true
for the
set_fact
task, which will define the variable and cause the
debug
conditional to evaluate to
false
.
If this is not the behavior you want, use an
include_*
statement to apply a condition only to that statement itself.
using a conditional on include_* only applies to the include task itself
main.yml
-
hosts
:
all
tasks
:
-
include_tasks
:
other_tasks.yml
note "include"
when
:
x is not defined
Now if
x
is initially undefined, the debug task will not be skipped because the conditional is evaluated at the time of the include and does not apply to the individual tasks.
You can apply conditions to

import_playbook
as well as to the other
import_*
statements. When you use this approach, Ansible returns a ?skipped? message for every task on every host that does not match the criteria, creating repetitive output. In many cases the
group_by module
can be a more streamlined way to accomplish the same objective; see
Handling OS and distro differences
.
Conditionals with includes
?
When you use a conditional on an
include_*
statement, the condition is applied only to the include task itself and not to any other tasks within the included file(s). To contrast with the example used for conditionals on imports above, look at the same playbook and tasks file, but using an include instead of an import:
Includes let you reuse a file to define a variable when it is not already defined
main.yml
-
include_tasks
:
other_tasks.yml
when
:
x is not defined
other_tasks.yml
-
name
:
Set a variable
ansible.builtin.set_fact
:
x
:
foo
-
name
:
Print a variable
ansible.builtin.debug
:
var
:
x
Ansible expands this at execution time to the equivalent of:
main.yml
-
include_tasks
:
other_tasks.yml
when
:
x is not defined
if condition is met, Ansible includes other_tasks.yml

```
# other_tasks.yml
-
name
:
Set a variable
ansible.builtin.set_fact
:
x
:
foo
# no condition applied to this task, Ansible sets the value of x to foo
-
name
:
Print a variable
ansible.builtin.debug
:
var
:
x
# no condition applied to this task, Ansible prints the debug statement
```

By using
include_tasks
instead of
import_tasks
, both tasks from
other_tasks.yml
will be executed as expected. For more information on the differences between
include
v
import
see
Reusing Ansible artifacts
.

Conditionals with roles
?

There are three ways to apply conditions to roles:

Add the same condition or conditions to all tasks in the role by placing your
when
statement under the
roles
keyword. See the example in this section.

Add the same condition or conditions to all tasks in the role by placing your
when
statement on a static
import_role
in your playbook.

Add a condition or conditions to individual tasks or blocks within the role itself. This is the only approach that allows you to select or skip some tasks within the role based on your
when
statement. To select or skip tasks within the role, you must have conditions set on individual tasks or blocks, use the dynamic
include_role
in your playbook, and add the condition or conditions to the include. When you use this approach, Ansible applies the

condition to the include itself plus any tasks in the role that also have that
when
statement.
When you incorporate a role in your playbook statically with the
roles
keyword, Ansible adds the conditions you define to all the tasks in the role. For example:
-
hosts
:
webservers
roles
:
-
role
:
debian_stock_config
when
:
ansible_facts['os_family'] == 'Debian'
Selecting variables, files, or templates based on facts
?
Sometimes the facts about a host determine the values you want to use for certain variables or even the file or template you want to select for that host. For example, the names of packages are different on CentOS and Debian. The configuration files for common services are also different on different OS flavors and versions. To load different variables files, templates, or other files based on a fact about the hosts:
name your vars files, templates, or files to match the Ansible fact that differentiates them
select the correct vars file, template, or file for each host with a variable based on that Ansible fact
Ansible separates variables from tasks, keeping your playbooks from turning into arbitrary code with nested conditionals. This approach results in more streamlined and auditable configuration rules because there are fewer decision points to track.
Selecting variables files based on facts
?
You can create a playbook that works on multiple platforms and OS versions with a minimum of syntax by placing your variable values in vars files and conditionally importing them. If you want to install Apache on some CentOS and some Debian servers, create variables files with YAML keys and values. For example:
---
for vars/RedHat.yml
apache
:
httpd
somethingelse
:
42
Then import those variables files based on the facts you gather on the hosts in your playbook:

-
hosts
:
webservers
remote_user
:
root
vars_files
:

-
"vars/common.yml"
-
[
"vars/{{
ansible_facts['os_family']
}}.yml"
,
"vars/os_defaults.yml"
]
tasks
:
-
name
:
Make sure apache is started
ansible.builtin.service
:
name
:
'{{
apache
}}'
state
:
started
Ansible gathers facts on the hosts in the webservers group, then interpolates the variable ?ansible_facts[?os_family?]? into a list of file names. If you have hosts with Red Hat operating systems (CentOS, for example), Ansible looks for ?vars/RedHat.yml?. If that file does not exist, Ansible attempts to load ?vars/os_defaults.yml?. For Debian hosts, Ansible first looks for ?vars/Debian.yml?, before falling back on ?vars/os_defaults.yml?. If no files in the list are found, Ansible raises an error.
Selecting files and templates based on facts
?
You can use the same approach when different OS flavors or versions require different configuration files or templates. Select the appropriate file or template based on the variables assigned to each host. This approach is often much cleaner than putting a lot of conditionals into a single template to cover multiple OS or package versions.
For example, you can template out a configuration file that is very different between, say, CentOS and Debian:
-
name
:
Template a file
ansible.builtin.template
:
src
:
"{{
item
}}"
dest
:
/etc/myapp/foo.conf
loop
:
"{{

query('first_found',
{
'files':
myfiles,
'paths':
mypaths})
}}"
vars
:
myfiles
:
-
"{{
ansible_facts['distribution']
}}.conf"
-
default.conf
mypaths
:
[
'search_location_one/somedir/'
,
'/opt/other_location/somedir/'
]
Debugging conditionals
?
If your conditional
when
statement is not behaving as you intended, you can add a
debug
statement to determine if the condition evaluates to
true
or
false
. A common cause of unexpected behavior in conditionals is testing an integer as a string or a string as an integer. To debug a conditional statement, add the entire statement as the
var:
value in a
debug
task. Ansible then shows the test and how the statement evaluates. For example, here is a set of tasks and sample output:
-
name
:
check value of return code
ansible.builtin.debug
:
var
:
bar_status.rc
-
name
:
check test for rc value as string

ansible.builtin.debug
:
var
:
bar_status.rc == "127"
-
name
:
check test for rc value as integer
ansible.builtin.debug
:
var
:
bar_status.rc == 127
TASK
[
check value of return code
]
****
ok
:
[
foo-1
]
=>
{
"bar_status.rc"
:
"127"
}
TASK
[
check test for rc value as string
]
****
ok
:
[
foo-1
]
=>
{
"bar_status.rc == \"127\""
:
false
}
TASK
[
check test for rc value as integer
]
****
ok
:
[

foo-1
]
=>
{
"bar_status.rc == 127"
:
true
}
Commonly-used facts
?
The following Ansible facts are frequently used in conditionals.
ansible_facts[?distribution?]
?
Possible values (sample, not complete list):
Alpine
Altlinux
Amazon
Archlinux
ClearLinux
Coreos
CentOS
Debian
Fedora
Gentoo
Mandriva
NA
OpenWrt
OracleLinux
RedHat
Slackware
SLES
SMGL
SUSE
Ubuntu
VMwareESX
ansible_facts[?distribution_major_version?]
?
The major version of the operating system. For example, the value is
16
for Ubuntu 16.04.
ansible_facts[?os_family?]
?
Possible values (sample, not complete list):
AIX
Alpine
Altlinux
Archlinux
Darwin
Debian
FreeBSD
Gentoo
HP-UX
Mandrake
RedHat

SMGL
Slackware
Solaris
Suse
Windows
See also
Working with playbooks
An introduction to playbooks
Roles
Playbook organization by roles
General tips
Tips and tricks for playbooks
Using variables
All about variables
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html

Ansible playbooks ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Ansible playbooks
Playbook syntax
Playbook execution
Task execution
Desired state and idempotency
Running playbooks
Running playbooks in check mode
Ansible-Pull
Verifying playbooks
ansible-lint
Working with playbooks
Executing playbooks
Advanced playbook syntax
Manipulating data
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started

Ansible playbooks

?

Ansible Playbooks provide a repeatable, reusable, simple configuration management and multimachine deployment system that is well suited to deploying complex applications. If you need to execute a task with Ansible more than once, you can write a playbook and put the playbook under source control. You can then use the playbook to push new configurations or confirm the configuration of remote systems.

Playbooks allow you to perform the following actions:

Declare configurations.

Orchestrate steps of any manual ordered process on multiple sets of machines in a defined order.

Launch tasks synchronously or
asynchronously
.

?
You express playbooks in YAML format with a minimum of syntax. If you are not familiar with YAML, review the
YAML Syntax
overview and consider installing an add-on for your text editor (see
Other Tools and Programs
) to help you write clean YAML syntax in your playbooks.
A playbook consists of one or more ?plays? in an ordered list. The terms ?playbook? and ?play? are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.
Playbook execution
?
A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple plays can orchestrate multimachine deployments, running one play on your webservers, another play on your database servers, and a third play on your network infrastructure. At a minimum, each play defines two things:
The managed nodes to target, using a
pattern
.
At least one task to execute.
For Ansible 2.10 and later, you should use the fully-qualified collection name (FQCN) in your playbooks. Using the FQCN ensures that you have selected the correct module, because multiple collections can contain modules with the same name. For example,
user
. See
Using collections in a playbook
.
In the following example, the first play targets the web servers and the second play targets the database servers.

-
name
:
Update web servers
hosts
:
webservers
remote_user
:
root
tasks
:
-
name
:
Ensure apache is at the latest version
ansible.builtin.yum
:
name
:
httpd
state
:
latest
-
name
:

Write the apache config file
ansible.builtin.template
:
src
:
/srv/httpd.j2
dest
:
/etc/httpd.conf
-
name
:
Update db servers
hosts
:
databases
remote_user
:
root
tasks
:
-
name
:
Ensure postgresql is at the latest version
ansible.builtin.yum
:
name
:
postgresql
state
:
latest
-
name
:
Ensure that postgresql is started
ansible.builtin.service
:
name
:
postgresql
state
:
started
Your playbook can include more than just a hosts line and tasks. For example, the playbook above sets a
remote_user
for each play. The
remote_user
is the user account for the SSH connection. You can add other
Playbook Keywords
at the playbook, play, or task level to influence how Ansible behaves. Playbook keywords can control the
connection plugin
, whether to use

privilege escalation
, how to handle errors, and more. To support a variety of environments, you can set many of these parameters as command-line flags in your Ansible configuration, or in your inventory. Learning the
precedence rules
for these sources of data helps you as you expand your Ansible ecosystem.
Task execution
?
By default, Ansible executes each task in order, one at a time, against all machines matched by the host pattern. Each task executes a module with specific arguments. After a task has executed on all target machines, Ansible moves to the next task. You can use
strategies
to change this default behavior. Within each play, Ansible applies the same task directives to all hosts. If a task fails on a host, Ansible removes that host from the rotation for the rest of the playbook.
When you run a playbook, Ansible returns information about connections, the
name
lines of all your plays and tasks, whether each task has succeeded or failed on each machine, and whether each task has made a change on each machine. At the bottom of the playbook execution, Ansible provides a summary of the nodes that were targeted and how they performed. General failures and fatal ?unreachable? communication attempts are kept separate in the counts.
Desired state and idempotency
?
Most Ansible modules check whether the desired final state has already been achieved and exit without performing any actions if that state has been achieved. Repeating the task does not change the final state. Modules that behave this way are ?idempotent?. Whether you run a playbook once or multiple times, the outcome should be the same. However, not all playbooks and not all modules behave this way. If you are unsure, test your playbooks in a sandbox environment before running them multiple times in production.
Running playbooks
?
To run your playbook, use the
ansible-playbook
command.
ansible-playbook
playbook.yml
-f
10
Use the
--verbose
flag when running your playbook to see detailed output from successful and unsuccessful tasks.
Running playbooks in check mode
?
The Ansible check mode allows you to execute a playbook without applying any alterations to your systems. You can use check mode to test playbooks before you implement them in a production environment.
To run a playbook in check mode, pass the
-C
or
--check
flag to the
ansible-playbook
command:
ansible-playbook
--check
playbook.yaml
Executing this command runs the playbook normally. Instead of implementing any modifications, Ansible provides a report on the changes it would have made. This report includes details such as file modifications, command execution,

and module calls.
Check mode offers a safe and practical approach to examine the functionality of your playbooks without risking unintended changes to your systems. Check mode is also a valuable tool for troubleshooting playbooks that are not functioning as expected.
Ansible-Pull
?
You can invert the Ansible architecture so that nodes check in to a central location instead of you pushing configuration out to them.
The
ansible-pull
command is a small script that checks out a repo of configuration instructions from git and then runs
ansible-playbook
against that content.
If you load balance your checkout location,
ansible-pull
scales infinitely.
Run
ansible-pull
--help
for details.
Verifying playbooks
?
You may want to verify your playbooks to catch syntax errors and other problems before you run them. The
ansible-playbook
command offers several options for verification, including
--check
,
--diff
,
--list-hosts
,
--list-tasks
, and
--syntax-check
. The
Tools for validating playbooks
topic describes other tools for validating and testing playbooks.
ansible-lint
?
You can use
ansible-lint
for detailed, Ansible-specific feedback on your playbooks before you execute them. For example, if you run
ansible-lint
on the playbook called
verify-apache.yml
near the top of this page, you should get the following results:
$
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
The
ansible-lint default rules
page describes each error.
See also
ansible-lint
Learn how to test Ansible Playbooks syntax
YAML Syntax
Learn about YAML syntax
General tips
Tips for managing playbooks in the real world
Collection Index
Browse existing collections, modules, and plugins
Should you develop a module?
Learn to extend Ansible by writing your own modules
Patterns: targeting hosts and groups
Learn about how to select hosts
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html

Using variables ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Ansible playbooks
Working with playbooks
Templating (Jinja2)
Using filters to manipulate data
Tests
Lookups
Python3 in templates
The now function: get the current time
The undef function: add hint for undefined variables
Loops
Controlling where tasks run: delegation and local actions
Conditionals
Blocks
Handlers: running operations on change
Error handling in playbooks
Setting the remote environment
Working with language-specific version managers
Reusing Ansible artifacts
Roles
Module defaults
Interactive input: prompts
Using variables
Creating valid variable names
Simple variables
When to quote variables (a YAML gotcha)
List variables
Dictionary variables
Combining variables
Registering variables
Referencing nested variables
Transforming variables with Jinja2 filters

Using Ansible playbooks
Working with playbooks
Using variables
Edit on GitHub
Using variables
?
Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple systems with a single command. To represent the variations among those different systems, you can create variables with standard YAML syntax, including lists and dictionaries. You can define these variables in your playbooks, in your
inventory
, in reusable
files
or
roles
, or at the command line. You can also create variables during a playbook run by registering the return value of a task as a new variable.
After you create a variable, you can use it in module arguments, in
conditional ?when? statements
, in
templates
, and in
loops
.
After you understand the concepts and examples on this page, read about
Ansible facts
, which are variables you retrieve from remote systems.
Creating valid variable names
Simple variables
Defining simple variables
Referencing simple variables
When to quote variables (a YAML gotcha)
List variables
Defining variables as lists
Referencing list variables
Dictionary variables
Defining variables as key-value dictionaries
Referencing key-value dictionary variables
Combining variables
Combining list variables
Combining dictionary variables
Using the merge_variables lookup
Registering variables
Referencing nested variables
Transforming variables with Jinja2 filters
Where to set variables
Defining variables in inventory
Defining variables in a play
Defining variables in included files and roles
Defining variables at runtime
Key-value format
JSON string format
Vars from a JSON or YAML file
Variable precedence: where should I put a variable?

Understanding variable precedence
Scoping variables
Tips on where to set variables
Using advanced variable syntax
Creating valid variable names
?
Not all strings are valid Ansible variable names. A variable name can only include letters, numbers, and underscores.
Python keywords
or
playbook keywords
are not valid variable names. A variable name cannot begin with a number.
Variable names can begin with an underscore. In many programming languages, variables that begin with an underscore are private. This is not true in Ansible. Ansible treats variables that begin with an underscore the same as any other variable. Do not rely on this convention for privacy or security.
This table gives examples of valid and invalid variable names:
Valid variable names
Not valid
foo
*foo
,
Python keywords
such as
async
and
lambda
foo_env
playbook keywords
such as
environment
foo_port
foo-port
,
foo
port
,
foo.port
foo5
,
_foo
5foo
,
12
Ansible defines certain
variables
internally. You cannot define these variables.
Avoid variable names that overwrite Jinja2 global functions listed in
Working with playbooks
, such as
lookup
,
query
,
q
,

now
, and
undef
.
Simple variables
?
Simple variables combine a variable name with a single value. You can use this syntax, and the syntax for lists and dictionaries shown below, in a variety of places. For details about setting variables in inventory, in playbooks, in reusable files, in roles, or at the command line, see
Where to set variables
.
Defining simple variables
?
You can define a simple variable using standard YAML syntax. For example:
remote_install_path: /opt/my_app_config
Referencing simple variables
?
After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces. For example, the expression
My
amp
goes
to
{{
max_amp_value
}}
demonstrates the most basic form of variable substitution. You can use Jinja2 syntax in playbooks. The following example shows a variable that defines the location of a file, which can vary from one system to another:
ansible.builtin.template
:
src
:
foo.cfg.j2
dest
:
'
{{
remote_install_path
}}
/foo.cfg'
Ansible allows Jinja2 loops and conditionals in
templates
but not in playbooks. You cannot create a loop of tasks. Ansible playbooks are pure machine-parseable YAML.
When to quote variables (a YAML gotcha)
?
If you start a value with
{{
foo
}}
, you must quote the whole expression to create valid YAML syntax. If you do not quote the whole expression, the YAML parser cannot interpret the syntax. The parser cannot determine if it is a variable or the start of a YAML dictionary. For guidance on writing YAML, see the
YAML Syntax
documentation.

If you use a variable without quotes, like this:
```
- hosts: app_servers
  vars:
    app_path: {{ base_path }}/22
```
You will see:
ERROR!
Syntax
Error
while
loading
YAML.
If you add quotes, Ansible works correctly:
-
hosts
:
app_servers
vars
:
app_path
:
"
{{
base_path
}}
/22"
List variables
?
A list variable combines a variable name with multiple values. You can store the multiple values as an itemized list or in square brackets
[]
, separated with commas.
Defining variables as lists
?
You can define variables with multiple values using YAML lists. For example:
region
:
-
northeast
-
southeast
-
midwest
Referencing list variables
?
If you use a variable defined as a list (also called an array), you can use individual, specific items from that list. The first item in a list is item 0, the second item is item 1, and so on. For example:
region
:
"
{{
region
[
0
]

}}
"
The value of this expression would be ?northeast?.
Dictionary variables
?
A dictionary stores data in key-value pairs. Usually, you use dictionaries to store related data, such as the information contained in an ID or a user profile.
Defining variables as key-value dictionaries
?
You can define more complex variables using YAML dictionaries. A YAML dictionary maps keys to values. For example:
foo
:
field1
:
one
field2
:
two
Referencing key-value dictionary variables
?
If you use a variable defined as a key-value dictionary (also called a hash), you can use individual, specific items from that dictionary using either bracket notation or dot notation:
foo['field1']
foo.field1
Both of these examples reference the same value (?one?). Bracket notation always works. Dot notation can cause problems because some keys collide with attributes and methods of python dictionaries. Use bracket notation if you use keys that start and end with two underscores, which are reserved for special meanings in python, or are any of the known public attributes:
add
,
append
,
as_integer_ratio
,
bit_length
,
capitalize
,
center
,
clear
,
conjugate
,
copy
,
count
,
decode
,
denominator
,
difference
,

difference_update
,
discard
,
encode
,
endswith
,
expandtabs
,
extend
,
find
,
format
,
fromhex
,
fromkeys
,
get
,
has_key
,
hex
,
imag
,
index
,
insert
,
intersection
,
intersection_update
,
isalnum
,
isalpha
,
isdecimal
,
isdigit
,
isdisjoint
,
is_integer
,
islower
,
isnumeric
,
isspace
,

issubset
,
issuperset
,
istitle
,
isupper
,
items
,
iteritems
,
iterkeys
,
itervalues
,
join
,
keys
,
ljust
,
lower
,
lstrip
,
numerator
,
partition
,
pop
,
popitem
,
real
,
remove
,
replace
,
reverse
,
rfind
,
rindex
,
rjust
,
rpartition
,
rsplit
,
rstrip
,

setdefault
,
sort
,
split
,
splitlines
,
startswith
,
strip
,
swapcase
,
symmetric_difference
,
symmetric_difference_update
,
title
,
translate
,
union
,
update
,
upper
,
values
,
viewitems
,
viewkeys
,
viewvalues
,
zfill
.
Combining variables
?
To merge variables that contain lists or dictionaries, you can use the following approaches.
Combining list variables
?
You can use the
set_fact
module to combine lists into a new
merged_list
variable as follows:
vars
:
list1
:
-
apple

-
banana
-
fig
list2
:
-
peach
-
plum
-
pear
tasks
:
-
name
:
Combine list1 and list2 into a merged_list var
ansible.builtin.set_fact
:
merged_list
:
"{{
list1
+
list2
}}"
Combining dictionary variables
?
To merge dictionaries, use the
combine
filter. For example:
vars
:
dict1
:
name
:
Leeroy Jenkins
age
:
25
occupation
:
Astronaut
dict2
:
location
:
Galway
country
:
Ireland
postcode

:
H71 1234
tasks
:
-
name
:
Combine dict1 and dict2 into a merged_dict var
ansible.builtin.set_fact
:
merged_dict
:
"{{
dict1
|
ansible.builtin.combine(dict2)
}}"
For more details, see
ansible.builtin.combine
.
Using the merge_variables lookup
?
To merge variables that match the given prefixes, suffixes, or regular expressions, you can use the
community.general.merge_variables
lookup. For example:
merged_variable
:
"{{
lookup('community.general.merge_variables',
'__my_pattern',
pattern_type='suffix')
}}"
For more details and example usage, refer to the
community.general.merge_variables lookup documentation
.
Registering variables
?
You can create a variable from the output of an Ansible task with the task keyword
register
. You can use the registered variable in any later task in your play. For example:
-
hosts
:
web_servers
tasks
:
-
name
:
Run a shell command and register its output as a variable
ansible.builtin.shell
:
/usr/bin/foo
register

:
foo_result
ignore_errors
:
true
-
name
:
Run a shell command using output of the previous task
ansible.builtin.shell
:
/usr/bin/bar
when
:
foo_result.rc == 5
For more examples of using registered variables in conditions on later tasks, see
Conditionals
. Registered variables may be simple variables, list variables, dictionary variables, or complex nested data structures. The documentation for each module includes a
RETURN
section that describes the return values for that module. To see the values for a particular task, run your playbook with
-v
.
Registered variables are stored in memory. You cannot cache registered variables for use in future playbook runs. A registered variable is valid only on the host for the rest of the current playbook run, including subsequent plays within the same playbook run.
Registered variables are host-level variables. When you register a variable in a task with a loop, the registered variable contains a value for each item in the loop. The data structure placed in the variable during the loop contains a
results
attribute, which is a list of all responses from the module. For a more in-depth example of how this works, see the
Loops
section on using register with a loop.
If a task fails or is skipped, Ansible still registers a variable with a failure or skipped status, unless the task is skipped based on tags. See
Tags
for information on adding and using tags.
Referencing nested variables
?
Many registered variables and
facts
are nested YAML or JSON data structures. You cannot access values from these nested data structures with the simple
{{
foo
}}
syntax. You must use either bracket notation or dot notation. For example, to reference an IP address from your facts using bracket notation:
'
{{
ansible_facts
[
"eth0"
][
"ipv4"
][

"address"
]
}}
'
To reference an IP address from your facts using dot notation:
{{
ansible_facts.eth0.ipv4.address
}}
Transforming variables with Jinja2 filters
?
Jinja2 filters let you transform the value of a variable within a template expression. For example, the
capitalize
filter capitalizes any value passed to it; the
to_yaml
and
to_json
filters change the format of your variable values. Jinja2 includes many
built-in filters
, and Ansible supplies many more filters. To find more examples of filters, see
Using filters to manipulate data
.
Where to set variables
?
You can define variables in a variety of places, such as in inventory, in playbooks, in reusable files, in roles, and at the command line. Ansible loads every possible variable it finds, then chooses the variable to apply based on
variable precedence rules
.
Defining variables in inventory
?
You can define different variables for each host individually, or set shared variables for a group of hosts in your inventory. For example, if all machines in the
[boston]
group use ?boston.ntp.example.com? as an NTP server, you can set a group variable. The
How to build your inventory
page has details on setting
host variables
and
group variables
in inventory.
Defining variables in a play
?
You can define variables directly in a playbook play:
-
hosts
:
webservers
vars
:
http_port
:
80
When you define variables in a play, they are visible only to tasks executed in that play.
Defining variables in included files and roles
?

You can define variables in reusable variables files or in reusable roles. If you define variables in reusable variable files, the sensitive variables are separated from playbooks. This separation enables you to store your playbooks in a source control software and even share the playbooks, without the risk of exposing passwords or other sensitive and personal data. For information about creating reusable files and roles, see
Reusing Ansible artifacts
.
This example shows how you can include variables defined in an external file:

-
hosts
:
all
remote_user
:
root
vars
:
favcolor
:
blue
vars_files
:
-
/vars/external_vars.yml
tasks
:
-
name
:
This is just a placeholder
ansible.builtin.command
:
/bin/echo foo
The contents of each variables file is a simple YAML dictionary. For example:
---
in the above example, this would be vars/external_vars.yml
somevar
:
somevalue
password
:
magic
You can keep per-host and per-group variables in similar files. To learn about organizing your variables, see
Organizing host and group variables
.
Defining variables at runtime
?
You can define variables when you run your playbook by passing variables at the command line using the
--extra-vars
(or
-e
) argument. You can also request user input with a
vars_prompt
(see

Interactive input: prompts
). If you pass variables at the command line, use a single quoted string that contains one or more variables in one of the formats below.
Key-value format
?
Values passed in using the
key=value
syntax are interpreted as strings. Use the JSON format if you need to pass non-string values such as Booleans, integers, floats, and lists.
ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
JSON string format
?
ansible-playbook
release.yml
--extra-vars
'{"version":"1.23.45","other_variable":"foo"}'
ansible-playbook
arcade.yml
--extra-vars
'{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
When passing variables with
--extra-vars
, you must escape quotes and other special characters appropriately for both your markup (for example, JSON) and for your shell:
ansible-playbook
arcade.yml
--extra-vars
"{\"name\":\"Conan O\'Brien\"}"
ansible-playbook
arcade.yml
--extra-vars
'{"name":"Conan O'
\\\'
'Brien"}'
ansible-playbook
script.yml
--extra-vars
"{\"dialog\":\"He said \\\"I just can\'t get enough of those single and double-quotes"
\!
"\\\"\"}"
Vars from a JSON or YAML file
?
If you have a lot of special characters, use a JSON or YAML file containing the variable definitions. Prepend both JSON and YAML file names with
@
.
ansible-playbook release.yml --extra-vars "@some_file.json"
ansible-playbook release.yml --extra-vars "@some_file.yaml"
Variable precedence: where should I put a variable?
?
You can set multiple variables with the same name in many different places. If you do this, Ansible loads every possible variable it finds, and then chooses the variable to apply based on variable precedence. In other words, the different variables will override each other in a certain order.
Teams and projects that agree on guidelines for defining variables (where to define certain types of variables) usually

avoid variable precedence concerns. You should define each variable in one place. Determine where to define a variable, and keep it simple. For examples, see
Tips on where to set variables
.
Some behavioral parameters that you can set in variables you can also set in Ansible configuration, as command-line options, and using playbook keywords. For example, you can define the user that Ansible uses to connect to remote devices as a variable with
ansible_user
, in a configuration file with
DEFAULT_REMOTE_USER
, as a command-line option with
-u
, and with the playbook keyword
remote_user
. If you define the same parameter in a variable and by another method, the variable overrides the other setting. This approach allows host-specific settings to override more general settings. For examples and more details on the precedence of these various settings, see
Controlling how Ansible behaves: precedence rules
.
Understanding variable precedence
?
Ansible does apply variable precedence, and you might have a use for it. Here is the order of precedence from least to greatest (the last listed variables override all other variables):
Command-line values (for example,
-u
my_user
, these are not variables)
Role defaults (as defined in
Role directory structure
)
[
1
]
Inventory file or script group vars
[
2
]
Inventory group_vars/all
[
3
]
Playbook group_vars/all
[
3
]
Inventory group_vars/*
[
3
]
Playbook group_vars/*
[
3
]
Inventory file or script host vars

[
2
]
Inventory host_vars/*
[
3
]
Playbook host_vars/*
[
3
]
Host facts and cached set_facts
[
4
]
Play vars
Play vars_prompt
Play vars_files
Role vars (as defined in
Role directory structure
)
Block vars (for tasks in block only)
Task vars (for the task only)
include_vars
Registered vars and set_facts
Role (and include_role) params
include params
Extra vars (for example,
-e
"user=my_user"
)(always win precedence)
In general, Ansible gives precedence to variables that were defined more recently, more actively, and with more explicit scope. Variables in the defaults folder inside a role are easily overridden. Anything in the vars directory of the role overrides previous versions of that variable in the namespace. Host or inventory variables override role defaults, but explicit includes such as the vars directory or an
include_vars
task override inventory variables.
Ansible merges different variables set in inventory so that more specific settings override more generic settings. For example,
ansible_ssh_user
specified as a group_var is overridden by
ansible_user
specified as a host_var. For details about the precedence of variables set in inventory, see
How variables are merged
.
Footnotes
[
1
]
Tasks in each role see their own role?s defaults. Tasks defined outside of a role see the last role?s defaults.
[
2
]
(

1
,
2
)
Variables defined in inventory file or provided by dynamic inventory.
[
3
]
(
1
,
2
,
3
,
4
,
5
,
6
)
Includes vars added by ?vars plugins? as well as host_vars and group_vars which are added by the default vars plugin shipped with Ansible.
[
4
]
When created with set_facts?s cacheable option, variables have the high precedence in the play,
but are the same as a host facts precedence when they come from the cache.
Note
Within any section, redefining a var overrides the previous instance. If multiple groups have the same variable, the last one loaded wins. If you define a variable twice in a play?s
vars:
section, the second one wins.
The previous text describes the default config
hash_behavior=replace
. Switch to
merge
to overwrite only partially.
Scoping variables
?
You can decide where to set a variable based on the scope you want that value to have. Ansible has three main scopes:
Global: this is set by config, environment variables and the command line
Play: each play and contained structures, vars entries (vars; vars_files; vars_prompt), role defaults and vars.
Host: variables directly associated to a host, like inventory, include_vars, facts or registered task outputs
Inside a template, you automatically have access to all variables that are in scope for a host, plus any registered variables, facts, and magic variables.
Tips on where to set variables
?
You should choose where to define a variable based on the kind of control you might want over values.
Set variables in inventory that deal with geography or behavior. Since groups are frequently the entity that maps roles to hosts, you can often set variables on the group instead of defining them on a role. Remember that child groups override parent groups, and host variables override group variables. See
Defining variables in inventory
for details on setting host and group variables.

Set common defaults in a
group_vars/all
file. See
Organizing host and group variables
for details on how to organize host and group variables in your inventory. You generally place group variables alongside your inventory file, but they can also be returned by dynamic inventory (see
Working with dynamic inventory
) or defined in AWX or on the
Red Hat Ansible Automation Platform
from the UI or API:

```
---
# file: /etc/ansible/group_vars/all
# this is the site wide default
ntp_server
:
default-time.example.com
```

Set location-specific variables in
group_vars/my_location
files. All groups are children of the
all
group, so variables set here override those set in
group_vars/all
:

```
---
# file: /etc/ansible/group_vars/boston
ntp_server
:
boston-time.example.com
```

If one host used a different NTP server, you could set that in a host_vars file, which would override the group variable:

```
---
# file: /etc/ansible/host_vars/xyz.boston.example.com
ntp_server
:
override.example.com
```

Set defaults in roles to avoid undefined-variable errors. If you share your roles, other users can rely on the reasonable defaults you added in the
roles/x/defaults/main.yml
file, or they can easily override those values in inventory or at the command line. See
Roles
for more info. For example:

```
---
# file: roles/x/defaults/main.yml
# if no other value is supplied in inventory or as a parameter, this value will be used
http_port
:
80
```

Set variables in roles to ensure a value is used in that role and is not overridden by inventory variables. If you are not sharing your role with others, you can define app-specific behaviors like ports this way, in
roles/x/vars/main.yml
. If you are sharing roles with others, putting variables here makes them harder to override, although they can still be overridden by passing a parameter to the role or setting a variable with
-e
:

```
---
```

```
# file: roles/x/vars/main.yml
# this will absolutely be used in this role
http_port
:
80
```

Pass variables as parameters when you call roles for maximum clarity, flexibility, and visibility. This approach overrides any defaults that exist for a role. For example:

```
roles
:
-
role
:
apache
vars
:
http_port
:
8080
```

When you read this playbook, it is clear that you have chosen to set a variable or override a default. You can also pass multiple values, which allows you to run the same role multiple times. See
Running a role multiple times in one play
for more details. For example:

```
roles
:
-
role
:
app_user
vars
:
myname
:
Ian
-
role
:
app_user
vars
:
myname
:
Terry
-
role
:
app_user
vars
:
myname
:
Graham
-
role
:
```

app_user
vars
:
myname
:
John
Variables set in one role are available to later roles. You can set variables in the role?s
vars
directory (as defined in
Role directory structure
) and use them in other roles and elsewhere in your playbook:
roles
:
-
role
:
common_settings
-
role
:
something
vars
:
foo
:
12
-
role
:
something_else
There are some protections in place to avoid the need to namespace variables. In this example, variables defined in ?common_settings? are available to ?something? and ?something_else? tasks, but tasks in ?something? have foo set at 12, even if ?common_settings? sets foo to 20.
Instead of worrying about variable precedence, we encourage you to think about how easily or how often you want to override a variable when deciding where to set it. If you are not sure what other variables are defined and you need a particular value, use
--extra-vars
(
-e
) to override all other variables.
Using advanced variable syntax
?
For information about advanced YAML syntax used to declare variables and have more control over the data placed in YAML files used by Ansible, see
Advanced playbook syntax
.
See also
Ansible playbooks
An introduction to playbooks
Conditionals
Conditional statements in playbooks
Using filters to manipulate data
Jinja2 filters and their uses
Loops

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

YAML Syntax ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
YAML Basics

YAML Syntax

?

This page provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

You may also wish to read Working with playbooks at the same time to see how this is used in practice.

YAML Basics

?

For Ansible, nearly every YAML file starts with a list.

Each item in the list is a list of key/value pairs, commonly called a ?hash? or a ?dictionary?. So, we need to know how to write lists and dictionaries in YAML.

There?s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with

```
---
```

and end with

```
...
```

. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a "- " (a dash and a space):

```
---
# A list of tasty fruits
-
Apple
-
Orange
-
Strawberry
```

-
Mango
...
A dictionary is represented in a simple
key:
value
form (the colon must be followed by a space):
An employee record
martin
:
name
:
Martin D'vloper
job
:
Developer
skill
:
Elite
More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:
Employee records
-
martin
:
name
:
Martin D'vloper
job
:
Developer
skills
:
-
python
-
perl
-
pascal
-
tabitha
:
name
:
Tabitha Bitumen
job
:
Developer
skills
:
-
lisp
-
fortran

-
erlang
Dictionaries and lists can also be represented in an abbreviated form if you really want to:
---
martin
:
{
name
:
Martin D'vloper
,
job
:
Developer
,
skill
:
Elite
}
fruits
:
[
'Apple'
,
'Orange'
,
'Strawberry'
,
'Mango'
]
These are called ?Flow collections?.
Ansible doesn?t really use these too much, but you can also specify a
boolean value
(true/false) in several forms:
create_key
:
true
needs_agent
:
false
knows_oop
:
True
likes_emacs
:
TRUE
uses_cvs
:
false
Use lowercase ?true? or ?false? for boolean values in dictionaries if you want to be compatible with default yamllint options.
Values can span multiple lines using
|
or

>
. Spanning multiple lines using a ?Literal Block Scalar?
|
will include the newlines and any trailing spaces.
Using a ?Folded Block Scalar?
>
will fold newlines to spaces; it is used to make what would otherwise be a very long line easier to read and edit.
In either case the indentation will be ignored.
Examples are:
include_newlines
:
|
exactly as you see
will appear these three
lines of poetry
fold_newlines
:
>
this is really a
single line of text
despite appearances
While in the above
>
example all newlines are folded into spaces, there are two ways to enforce a newline to be kept:
fold_some_newlines
:
>
a
b
c
d
e
f
Alternatively, it can be enforced by including newline
\n
characters:
fold_same_newlines
:
"a
b\nc
d\n
e\nf\n"
Let?s combine what we learned so far in an arbitrary YAML example.
This really has nothing to do with Ansible, but will give you a feel for the format:

An employee record
name
:
Martin D'vloper
job
:
Developer
skill
:

Elite
employed
:
True
foods
:
-
Apple
-
Orange
-
Strawberry
-
Mango
languages
:
perl
:
Elite
python
:
Elite
pascal
:
Lame
education
:
|
4 GCSEs
3 A-Levels
BSc in the Internet of Things
That?s all you really need to know about YAML to start writing
Ansible
playbooks.
Gotchas
?
While you can put just about anything into an unquoted scalar, there are some exceptions.
A colon followed by a space (or newline)
":
"
is an indicator for a mapping.
A space followed by the pound sign
"
#"
starts a comment.
Because of this, the following is going to result in a YAML syntax error:
foo: somebody said I should put a colon here: so I did

windows_drive: c:
?but this will work:
windows_path
:
c:\windows
You will want to quote hash values using colons followed by a space or the end of the line:

foo
:
'somebody
said
I
should
put
a
colon
here:
so
I
did'
windows_drive
:
'c:'
?and then the colon will be preserved.
Alternatively, you can use double quotes:
foo
:
"somebody
said
I
should
put
a
colon
here:
so
I
did"
windows_drive
:
"c:"
The difference between single quotes and double quotes is that in double quotes
you can use escapes:
foo
:
"a
\t
TAB
and
a
\n
NEWLINE"
The list of allowed escapes can be found in the YAML Specification under ?Escape Sequences? (YAML 1.1) or ?Escape Characters? (YAML 1.2).
The following is invalid YAML:
foo: "an escaped \' single quote"
Further, Ansible uses ?{{ var }}? for variables. If a value after a colon starts
with a ?{?, YAML will think it is a dictionary, so you must quote it, like so:
foo
:
"{{

variable
}}"
If your value starts with a quote the entire value must be quoted, not just part of it. Here are some additional examples of how to properly quote things:
foo
:
"{{
variable
}}/additional/string/literal"
foo2
:
"{{
variable
}}\\backslashes\\are\\also\\special\\characters"
foo3
:
"even
if
it
is
just
a
string
literal
it
must
all
be
quoted"
Not valid:
foo: "E:\\path\\"rest\\of\\path
In addition to
'
and
"
there are a number of characters that are special (or reserved) and cannot be used
as the first character of an unquoted scalar:
[]
{}
>
|
*
&
!
%
#
`
@
,
.
You should also be aware of
?
:
-

. In YAML, they are allowed at the beginning of a string if a non-space
character follows, but YAML processor implementations differ, so it is better to use quotes.
In Flow Collections, the rules are a bit more strict:
a scalar in block mapping: this } is [all , valid

flow mapping: { key: "you { should [use , quotes here" }
Boolean conversion is helpful, but this can be a problem when you want a literal
yes
or other boolean values as a string.
In these cases just use quotes:
non_boolean
:
"yes"
other_string
:
"False"
YAML converts certain strings into floating-point values, such as the string
1.0
. If you need to specify a version number (in a requirements.yml file, for
example), you will need to quote the value if it looks like a floating-point
value:
version
:
"1.0"
See also
Working with playbooks
Learn what playbooks can do and how to write/run them.
YAMLLint
YAML Lint (online) helps you debug YAML syntax if you are having problems
Wikipedia YAML syntax reference
A good guide to YAML syntax
YAML 1.1 Specification
The Specification for YAML 1.1, which PyYAML and libyaml are currently
implementing
YAML 1.2 Specification
For completeness, YAML 1.2 is the successor of 1.1
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/archive/ansible/2.3/

How do I handle different machines needing different user accounts or ports to log in with?
How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?
How do I configure a jump host to access servers that I have no direct access to?
How do I speed up management inside EC2?
How do I handle python pathing not having a Python 2.X in /usr/bin/python on a remote machine?
What is the best way to make content reusable/redistributable?
Where does the configuration file live and what can I configure in it?
How do I disable cowsay?
How do I see a list of all of the ansible_ variables?
How do I see all the inventory vars defined for my host?
How do I loop over a list of hosts in a group, inside of a template?
How do I access a variable name programmatically?
How do I access a variable of the first host in a group?
How do I copy files recursively onto a target host?
How do I access shell environment variables?
How do I generate crypted passwords for the user module?
Can I get training on Ansible?
Is there a web interface / REST API / etc?
How do I submit a change to the documentation?
How do I keep secret data in my playbook?
When should I use {{ }}? Also, how to interpolate variables or dynamic variable names
Why don?t you ship in X format?
I don?t see my question here
Glossary
YAML Syntax
YAML Basics
Gotchas
Ansible Porting Guides
Python 3 Support
Testing Python 3 with commands and playbooks
Testing Python 3 module support
Release and maintenance
AnsibleFest
Products
Community
Webinars & Training
Blog
Ansible Documentation
Docs
»
Ansible Documentation
You are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible documentation
.
Ansible Documentation
¶
About Ansible
¶
Welcome to the Ansible documentation!
Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
Ansible?s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a

minimum of moving parts, usage of OpenSSH for transport (with an accelerated socket mode and pull modes as alternatives), and a language that is designed around auditability by humans?even those not familiar with the program.
We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.
Ansible manages machines in an agent-less manner. There is never a question of how to
upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled.
Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized?it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.
This documentation covers the current released version of Ansible (2.2) and also some development version features (2.3). For recent features, we note in each section the version of Ansible where the feature was added.
Ansible, Inc. releases a new major release of Ansible approximately every two months. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, typically adding 20 or so new modules in each release.
Introduction
Quickstart Video
Playbooks
Playbooks: Special Topics
About Modules
Module Index
Detailed Guides
Developer Information
Community Information
Ansible Tower
Ansible Galaxy
Testing Strategies
Frequently Asked Questions
Glossary
YAML Syntax
Ansible Porting Guides
Python 3 Support
Release and maintenance
Next

Last updated on Sep 01, 2022.
Ansible docs are generated from
GitHub sources
using
Sphinx
using a theme provided by
Read the Docs
.

Content from: https://docs.ansible.com/archive/ansible/2.5/

Do-Until Loops
Using register with a loop
Looping over the inventory
Loop Control
Loops and Includes in 2.0
Blocks
Error Handling
Advanced Playbooks Features
Understanding Privilege Escalation
Asynchronous Actions and Polling
Check Mode (?Dry Run?)
Playbook Debugger
Delegation, Rolling Updates, and Local Actions
Setting the Environment (and Working With Proxies)
Working With Language-Specific Version Managers
Error Handling In Playbooks
Advanced Syntax
Working With Plugins
Prompts
Tags
Using Vault in playbooks
Start and Step
Playbook Keywords
Lookups
Strategies
Strategy Plugins
Best Practices
Content Organization
Staging vs Production
Rolling Updates
Always Mention The State
Group By Roles
Operating System and Distribution Variance
Bundling Ansible Modules With Playbooks
Whitespace and Comments
Always Name Tasks
Keep It Simple
Version Control
Variables and Vaults
Understanding Privilege Escalation
Become
Directives
Connection variables
Command line options
For those from Pre 1.9 , sudo and su still work!
Limitations
Become and Networks
network_cli and become
Setting enable mode for all tasks
authorize and auth_pass
Become and Windows
Administrative Rights
Local Service Accounts
Accounts without a Password

Porting Controller Code to Python 3
Controller String Strategy
Tips, tricks, and idioms to adopt
Porting Modules to Python 3
Module String Strategy
Tips, tricks, and idioms to adopt
Porting module_utils code to Python 3
Module_utils String Strategy
Python API
Python API example
Rebasing a Pull Request
Configuring Your Remotes
Rebasing Your Branch
Updating Your Pull Request
Getting Help Rebasing
Testing Ansible
Introduction
Types of tests
Testing within GitHub & Shippable
Organization
Rerunning a failing CI job
How to test a PR
Setup: Checking out a Pull Request
Testing the Pull Request
Want to know more about testing?
Repo Merge
Background
Why Are We Doing This (Again?)?
Metadata - Support/Ownership and Module Status
Release and maintenance
Committers Guidelines (for people with commit rights to Ansible on GitHub)
Features, High Level Design, and Roadmap
Our Workflow on GitHub
Addendum to workflow for Committers:
Roles on Core
General Rules
People
Ansible Style Guide
Ansible Style Guide
1. Why Use a Style Guide?
2. Resources
3. Basic Rules
3.1. Use Standard American English
3.2. Write for a Global Audience
3.3. Follow Naming Conventions
3.4. Important Information First
3.5. Sentence Structure
3.6. Avoid padding
3.7. Avoid redundant prepositional phrases
3.8. Avoid verbosity
3.9. Avoid pomposity
3.10. Action verbs, menus, and commands
4. Voice Style
4.1. Active Voice

7.46. Read-only
7.47. Real time/real-time
7.48. Refer to
7.49. See
7.50. Since
7.51. Tells
7.52. That/which
7.53. Then/than
7.54. Third-party
7.55. Troubleshoot
7.56. UK
7.57. UNIX®
7.58. Unset
7.59. US
7.60. User
7.61. Username
7.62. View
7.63. Within
7.64. World Wide Web
7.65. Webpage
7.66. Web server
7.67. Website
7.68. Who/whom
7.69. Will
7.70. Wish
7.71. x86
7.72. x86_64
7.73. You
7.74. You may
Scenario Guides
Cisco ACI Guide
What is Cisco ACI ?
Application Centric Infrastructure (ACI)
Application Policy Infrastructure Controller (APIC)
ACI Fabric
More information
Using the ACI modules
Querying ACI configuration
Common parameters
Proxy support
Return values
More information
ACI authentication
Password-based authentication
Signature-based authentication using certificates
Generate certificate and private key
Configure your local user
Use signature-based authentication with Ansible
More information
Using ACI REST with Ansible
Built-in idempotency
Using the aci_rest module
More information
Operational examples

K8S
Openshift
Commands modules
Crypto modules
Database modules
Influxdb
Misc
Mongodb
Mssql
Mysql
Postgresql
Proxysql
Vertica
Files modules
Identity modules
Cyberark
Ipa
Keycloak
Opendj
Inventory modules
Messaging modules
Monitoring modules
Zabbix
Net Tools modules
Basics
Exoscale
Infinity
Ldap
Nios
Network modules
A10
Aci
Aireos
Aos
Aruba
Asa
Avi
Bigswitch
Citrix
Cloudengine
Cloudvision
Cnos
Cumulus
Dellos10
Dellos6
Dellos9
Edgeos
Enos
Eos
F5
Fortimanager
Fortios
Illumos
Interface

Ios
Iosxr
Ironware
Junos
Layer2
Layer3
Netact
Netconf
Netscaler
Netvisor
Nso
Nuage
Nxos
Onyx
Ordnance
Ovs
Panos
Protocol
Radware
Routing
Sros
System
Vyos
Notification modules
Packaging modules
Language
Os
Remote Management modules
Foreman
Hpilo
Imc
Ipmi
Manageiq
Oneview
Stacki
Ucs
Source Control modules
Storage modules
Infinidat
Netapp
Purestorage
Zfs
System modules
Utilities modules
Helper
Logic
Web Infrastructure modules
Ansible_Tower
Windows modules
Playbook Keywords
Play
Role
Block
Task

ANSIBLE_COW_PATH
ANSIBLE_COW_SELECTION
ANSIBLE_COW_WHITELIST
ANSIBLE_FORCE_COLOR
ANSIBLE_NOCOLOR
ANSIBLE_NOCOWS
ANSIBLE_PIPELINING
ANSIBLE_SSH_ARGS
ANSIBLE_SSH_CONTROL_PATH
ANSIBLE_SSH_CONTROL_PATH_DIR
ANSIBLE_SSH_EXECUTABLE
ANSIBLE_SSH_RETRIES
ANY_ERRORS_FATAL
BECOME_ALLOW_SAME_USER
CACHE_PLUGIN
CACHE_PLUGIN_CONNECTION
CACHE_PLUGIN_PREFIX
CACHE_PLUGIN_TIMEOUT
COLOR_CHANGED
COLOR_DEBUG
COLOR_DEPRECATE
COLOR_DIFF_ADD
COLOR_DIFF_LINES
COLOR_DIFF_REMOVE
COLOR_ERROR
COLOR_HIGHLIGHT
COLOR_OK
COLOR_SKIP
COLOR_UNREACHABLE
COLOR_VERBOSE
COLOR_WARN
COMMAND_WARNINGS
DEFAULT_ACTION_PLUGIN_PATH
DEFAULT_ALLOW_UNSAFE_LOOKUPS
DEFAULT_ASK_PASS
DEFAULT_ASK_SU_PASS
DEFAULT_ASK_SUDO_PASS
DEFAULT_ASK_VAULT_PASS
DEFAULT_BECOME
DEFAULT_BECOME_ASK_PASS
DEFAULT_BECOME_EXE
DEFAULT_BECOME_FLAGS
DEFAULT_BECOME_METHOD
DEFAULT_BECOME_USER
DEFAULT_CACHE_PLUGIN_PATH
DEFAULT_CALLABLE_WHITELIST
DEFAULT_CALLBACK_PLUGIN_PATH
DEFAULT_CALLBACK_WHITELIST
DEFAULT_CONNECTION_PLUGIN_PATH
DEFAULT_DEBUG
DEFAULT_EXECUTABLE
DEFAULT_FACT_PATH
DEFAULT_FILTER_PLUGIN_PATH
DEFAULT_FORCE_HANDLERS

DEFAULT_FORKS
DEFAULT_GATHER_SUBSET
DEFAULT_GATHER_TIMEOUT
DEFAULT_GATHERING
DEFAULT_HANDLER_INCLUDES_STATIC
DEFAULT_HASH_BEHAVIOUR
DEFAULT_HOST_LIST
DEFAULT_INTERNAL_POLL_INTERVAL
DEFAULT_INVENTORY_PLUGIN_PATH
DEFAULT_JINJA2_EXTENSIONS
DEFAULT_KEEP_REMOTE_FILES
DEFAULT_LIBVIRT_LXC_NOSECLABEL
DEFAULT_LOAD_CALLBACK_PLUGINS
DEFAULT_LOCAL_TMP
DEFAULT_LOG_FILTER
DEFAULT_LOG_PATH
DEFAULT_LOOKUP_PLUGIN_PATH
DEFAULT_MANAGED_STR
DEFAULT_MODULE_ARGS
DEFAULT_MODULE_COMPRESSION
DEFAULT_MODULE_LANG
DEFAULT_MODULE_NAME
DEFAULT_MODULE_PATH
DEFAULT_MODULE_SET_LOCALE
DEFAULT_MODULE_UTILS_PATH
DEFAULT_NO_LOG
DEFAULT_NO_TARGET_SYSLOG
DEFAULT_NULL_REPRESENTATION
DEFAULT_POLL_INTERVAL
DEFAULT_PRIVATE_KEY_FILE
DEFAULT_PRIVATE_ROLE_VARS
DEFAULT_REMOTE_PORT
DEFAULT_REMOTE_USER
DEFAULT_ROLES_PATH
DEFAULT_SCP_IF_SSH
DEFAULT_SELINUX_SPECIAL_FS
DEFAULT_SFTP_BATCH_MODE
DEFAULT_SQUASH_ACTIONS
DEFAULT_SSH_TRANSFER_METHOD
DEFAULT_STDOUT_CALLBACK
DEFAULT_STRATEGY
DEFAULT_STRATEGY_PLUGIN_PATH
DEFAULT_SU
DEFAULT_SU_EXE
DEFAULT_SU_FLAGS
DEFAULT_SU_USER
DEFAULT_SUDO
DEFAULT_SUDO_EXE
DEFAULT_SUDO_FLAGS
DEFAULT_SUDO_USER
DEFAULT_SYSLOG_FACILITY
DEFAULT_TASK_INCLUDES_STATIC
DEFAULT_TEST_PLUGIN_PATH
DEFAULT_TIMEOUT

DEFAULT_TRANSPORT
DEFAULT_UNDEFINED_VAR_BEHAVIOR
DEFAULT_VARS_PLUGIN_PATH
DEFAULT_VAULT_ENCRYPT_IDENTITY
DEFAULT_VAULT_ID_MATCH
DEFAULT_VAULT_IDENTITY
DEFAULT_VAULT_IDENTITY_LIST
DEFAULT_VAULT_PASSWORD_FILE
DEFAULT_VERBOSITY
DEPRECATION_WARNINGS
DIFF_ALWAYS
DIFF_CONTEXT
DISPLAY_ARGS_TO_STDOUT
DISPLAY_SKIPPED_HOSTS
ENABLE_TASK_DEBUGGER
ERROR_ON_MISSING_HANDLER
GALAXY_IGNORE_CERTS
GALAXY_ROLE_SKELETON
GALAXY_ROLE_SKELETON_IGNORE
GALAXY_SERVER
GALAXY_TOKEN
HOST_KEY_CHECKING
INJECT_FACTS_AS_VARS
INVENTORY_ENABLED
INVENTORY_EXPORT
INVENTORY_IGNORE_EXTS
INVENTORY_IGNORE_PATTERNS
INVENTORY_UNPARSED_IS_FAILED
MAX_FILE_SIZE_FOR_DIFF
MERGE_MULTIPLE_CLI_TAGS
NETWORK_GROUP_MODULES
PARAMIKO_HOST_KEY_AUTO_ADD
PARAMIKO_LOOK_FOR_KEYS
PERSISTENT_COMMAND_TIMEOUT
PERSISTENT_CONNECT_RETRY_TIMEOUT
PERSISTENT_CONNECT_TIMEOUT
PERSISTENT_CONTROL_PATH_DIR
PLAYBOOK_VARS_ROOT
PLUGIN_FILTERS_CFG
RETRY_FILES_ENABLED
RETRY_FILES_SAVE_PATH
SHOW_CUSTOM_STATS
STRING_TYPE_FILTERS
SYSTEM_WARNINGS
TAGS_RUN
TAGS_SKIP
USE_PERSISTENT_CONNECTIONS
VARIABLE_PRECEDENCE
YAML_FILENAME_EXTENSIONS
Environment Variables
YAML Syntax
YAML Basics
Gotchas
Python 3 Support

On the controller side
Using Python 3 on the managed machines with commands and playbooks
What to do if an incompatibility is found
Release and maintenance
Testing Strategies
Integrating Testing With Ansible Playbooks
The Right Level of Testing
Check Mode As A Drift Test
Modules That Are Useful for Testing
Testing Lifecycle
Integrating Testing With Rolling Updates
Achieving Continuous Deployment
Conclusion
Sanity Tests
Sanity Tests » ansible-doc
Sanity Tests » azure-requirements
Sanity Tests » boilerplate
Sanity Tests » compile
Sanity Tests » configure-remoting-ps1
Sanity Tests » empty-init
Sanity Tests » import
Sanity Tests » integration-aliases
Groups
Requirements
Skipping
Unstable
Disabled
Unsupported
Cloud
Untested
Issues
Questions
Sanity Tests » line-endings
Sanity Tests » no-assert
Sanity Tests » no-basestring
Sanity Tests » no-dict-iteritems
Sanity Tests » no-dict-iterkeys
Sanity Tests » no-dict-itervalues
Sanity Tests » no-get-exception
Sanity Tests » no-illegal-filenames
Illegal Characters
Illegal Names
Sanity Tests » no-smart-quotes
Sanity Tests » no-tests-as-filters
Sanity Tests » no-underscore-variable
Sanity Tests » no-unicode_literals
Sanity Tests » pep8
Sanity Tests » pslint
Sanity Tests » pylint
Sanity Tests » replace-urlopen
Sanity Tests » required-and-default-attributes
Sanity Tests » rstcheck
Sanity Tests » sanity-docs
Sanity Tests » shebang

Docker
Extras split from Core
Tweaks/Fixes
AWS
Google
OpenStack
Azure load balancer
VMware
Windows
Network
Role revamp
Vault
Python3
Infrastructure Buildout and Changes
Ansible 2.3
General Comments from the Core Team
Repo Merge
Metadata
Documentation
Windows
Azure
Networking
Python3
Testing and CI
Amazon
Plugin Loader
ansible-ssh
Ansible 2.4
Administrivia and Process
Python 2.4 and 2.5 support discontinuation
Python 3
Ansible-Config
Inventory
Facts
PluginLoader
Static Loop Keyword
Vault
Globalize Callbacks
Plugins
Group Priorities
Runtime Check on Modules for Blacklisting
Disambiguate Includes
Windows
AWS
Azure
Google Cloud Platform
Network Roadmap
Contributor Quality of Life
Ansible 2.5
Engine improvements
Ansible Content Management
Ansible-Config
Inventory
Facts

PluginLoader
Static Loop Keyword
Vault
Role Versioning
Globalize Callbacks
Runtime Check on Modules for Blacklisting
Windows
General Cloud
AWS
Azure
Network Roadmap
Documentation
Contributor Quality of Life
Ansible Documentation
Docs
»
Ansible Documentation
You are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible documentation
.
Ansible Documentation
¶
About Ansible
¶
Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
Ansible?s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans?even those not familiar with the program.
We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.
Ansible manages machines in an agent-less manner. There is never a question of how to
upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled.
Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized?it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.
This documentation covers the current released version of Ansible (2.5) and also some development version features.
For recent features, we note in each section the version of Ansible where the feature was added.
Ansible releases a new major release of Ansible approximately every two months. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, adding many new modules in each release.
Installation, Upgrade & Configuration
Installation Guide
Basics / What Will Be Installed
What Version To Pick?
Control Machine Requirements
Managed Node Requirements
Installing the Control Machine
Ansible on GitHub
Configuring Ansible
Configuration file

The Website
The command line tool
Return Values
Common
Internal use
Ansible Configuration Settings
The configuration file
Common Options
Environment Variables
YAML Syntax
YAML Basics
Gotchas
Python 3 Support
On the controller side
Using Python 3 on the managed machines with commands and playbooks
What to do if an incompatibility is found
Release and maintenance
Testing Strategies
Integrating Testing With Ansible Playbooks
The Right Level of Testing
Check Mode As A Drift Test
Modules That Are Useful for Testing
Testing Lifecycle
Integrating Testing With Rolling Updates
Achieving Continuous Deployment
Conclusion
Sanity Tests
Sanity Tests » ansible-doc
Sanity Tests » azure-requirements
Sanity Tests » boilerplate
Sanity Tests » compile
Sanity Tests » configure-remoting-ps1
Sanity Tests » empty-init
Sanity Tests » import
Sanity Tests » integration-aliases
Sanity Tests » line-endings
Sanity Tests » no-assert
Sanity Tests » no-basestring
Sanity Tests » no-dict-iteritems
Sanity Tests » no-dict-iterkeys
Sanity Tests » no-dict-itervalues
Sanity Tests » no-get-exception
Sanity Tests » no-illegal-filenames
Sanity Tests » no-smart-quotes
Sanity Tests » no-tests-as-filters
Sanity Tests » no-underscore-variable
Sanity Tests » no-unicode_literals
Sanity Tests » pep8
Sanity Tests » pslint
Sanity Tests » pylint
Sanity Tests » replace-urlopen
Sanity Tests » required-and-default-attributes
Sanity Tests » rstcheck
Sanity Tests » sanity-docs

Last updated on Nov 29, 2022.

Content from: https://docs.ansible.com/archive/ansible/2.6/

Ansible Documentation ? Ansible Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.6
Installation, Upgrade & Configuration
Installation Guide
Configuring Ansible
Ansible Porting Guides
Using Ansible
User Guide
Contributing to Ansible
Ansible Community Guide
Extending Ansible
Developer Guide
Scenario Guides
Cisco ACI Guide
Amazon Web Services Guide
Microsoft Azure Guide
CloudStack Cloud Guide
Getting Started with Docker
Google Cloud Platform Guide
Using Ansible with the Packet host
Rackspace Cloud Guide
Continuous Delivery and Rolling Upgrades
Using Vagrant and Ansible
Ansible for VMWare
Ansible for VMware
Ansible for Network Automation
Ansible for Network Automation
Reference & Appendices
Module Index
Playbook Keywords
Ansible Galaxy
Return Values
Ansible Configuration Settings
YAML Syntax
Python 3 Support
Release and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Tower
Roadmaps
Ansible Roadmap
Ansible Documentation
Docs

»
Ansible Documentation
You are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible documentation
.
Ansible Documentation
¶
About Ansible
¶
Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
Ansible?s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans?even those not familiar with the program.
We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.
Ansible manages machines in an agent-less manner. There is never a question of how to
upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled.
Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced.
Ansible is decentralized?it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.
This documentation covers the current released version of Ansible (2.5) and also some development version features.
For recent features, we note in each section the version of Ansible where the feature was added.
Ansible releases a new major release of Ansible approximately every two months. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, adding many new modules in each release.
Installation, Upgrade & Configuration
Installation Guide
Basics / What Will Be Installed
What Version To Pick?
Control Machine Requirements
Managed Node Requirements
Installing the Control Machine
Ansible on GitHub
Configuring Ansible
Configuration file
Environmental configuration
Command line options
Ansible Porting Guides
Using Ansible
User Guide
Ansible Quickstart
Getting Started
Working with Command Line Tools
Introduction To Ad-Hoc Commands
Working with Inventory
Working With Dynamic Inventory
Working With Playbooks
Understanding Privilege Escalation
Ansible Vault
Working with Patterns

YAML Basics
Gotchas
Python 3 Support
On the controller side
Using Python 3 on the managed machines with commands and playbooks
What to do if an incompatibility is found
Release and maintenance
Testing Strategies
Integrating Testing With Ansible Playbooks
The Right Level of Testing
Check Mode As A Drift Test
Modules That Are Useful for Testing
Testing Lifecycle
Integrating Testing With Rolling Updates
Achieving Continuous Deployment
Conclusion
Sanity Tests
ansible-doc
azure-requirements
boilerplate
changelog
compile
configure-remoting-ps1
empty-init
import
integration-aliases
line-endings
no-assert
no-basestring
no-dict-iteritems
no-dict-iterkeys
no-dict-itervalues
no-get-exception
no-illegal-filenames
no-smart-quotes
no-tests-as-filters
no-underscore-variable
no-unicode_literals
pep8
pslint
pylint
replace-urlopen
required-and-default-attributes
rstcheck
sanity-docs
shebang
shellcheck
symlinks
test-constraints
use-argspec-type-path
use-compat-six
validate-modules
yamllint
Frequently Asked Questions

Next
Last updated on Jan 30, 2023.

Content from: https://docs.ansible.com/archive/ansible/2.7/

Ansible Documentation ? Ansible Documentation
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
Ansible
2.7
Installation, Upgrade & Configuration
Installation Guide
Configuring Ansible
Ansible Porting Guides
Using Ansible
User Guide
Contributing to Ansible
Ansible Community Guide
Extending Ansible
Developer Guide
Scenario Guides
Cisco ACI Guide
Amazon Web Services Guide
Microsoft Azure Guide
CloudStack Cloud Guide
Getting Started with Docker
Google Cloud Platform Guide
Infoblox Guide
Getting Started with Kubernetes and OpenShift
Cisco Meraki Guide
Using Ansible with the Packet host
Rackspace Cloud Guide
Continuous Delivery and Rolling Upgrades
Using Vagrant and Ansible
Vultr Guide
Ansible for VMWare
Ansible for VMware
Ansible for Network Automation
Ansible for Network Automation
Reference & Appendices
Module Index
Playbook Keywords
Ansible Galaxy
Return Values
Ansible Configuration Settings
YAML Syntax
Python 3 Support
Release and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities

Special Variables
Ansible Tower
Roadmaps
Ansible Roadmap
Ansible
Docs
»
Ansible Documentation
You are reading an unmaintained version of the Ansible documentation. Unmaintained Ansible versions can contain unfixed security vulnerabilities (CVE). Please upgrade to a maintained version. See
the latest Ansible documentation
.
Ansible Documentation
¶
About Ansible
¶
Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
Ansible?s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans?even those not familiar with the program.
We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.
Ansible manages machines in an agent-less manner. There is never a question of how to
upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled.
Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized?it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.
This documentation covers the version of Ansible noted in the upper left corner of this page. We maintain multiple versions of Ansible and of the documentation, so please be sure you are using the version of the documentation that covers the version of Ansible you?re using. For recent features, we note the version of Ansible where the feature was added.
Ansible releases a new major release of Ansible approximately three to four times per year. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. However, the community around new modules and plugins being developed and contributed moves very quickly, adding many new modules in each release.
Installation, Upgrade & Configuration
Installation Guide
Basics / What Will Be Installed
What Version To Pick?
Control Machine Requirements
Managed Node Requirements
Installing the Control Machine
Ansible on GitHub
Configuring Ansible
Configuration file
Environmental configuration
Command line options
Ansible Porting Guides
Using Ansible
User Guide
Ansible Quickstart

Last updated on Mar 31, 2023.

Content from: https://docs.ansible.com/automation-controller/

Index of /automation-controller
Index of /automation-controller
Name
Last modified
Size
Description
Parent Directory
-
4.0.0/
2022-12-09 03:20
-
4.0.1/
2022-12-09 04:04
-
4.1.0/
2025-10-18 22:24
-
4.1.1/
2025-10-18 22:55
-
4.1.2/
2025-10-19 22:03
-
4.1.3/
2025-10-18 22:13
-
4.1.4/
2025-10-18 22:35
-
4.2.0/
2025-10-18 22:39
-
4.2.1/
2025-10-18 22:52
-
4.2.2/
2025-10-18 22:18
-
4.3.0_old/
2022-12-16 03:15
-
4.3/
2025-10-18 22:33
-
4.4.0/
2022-11-30 18:41
-
4.4/
2025-10-18 22:49
-
4.5/
2025-10-19 22:01

-
4.6/
2025-10-18 22:05
-
4.7/
2025-09-10 05:28
-
latest/
2025-10-18 22:05
-

Content from: https://docs.ansible.com/automation-controller/4.1.2/

Index of /automation-controller/4.1.2
Index of /automation-controller/4.1.2
Name
Last modified
Size
Description
Parent Directory
-
html/
2025-10-18 22:02
-
html_ja/
2022-04-21 21:33
-
html_zh/
2022-04-21 21:46
-
pdf/
2023-02-10 05:45
-

Content from: https://docs.ansible.com/automation-controller/4.1.2/html/

Index of /automation-controller/4.1.2/html
Index of /automation-controller/4.1.2/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:02
-
controllerapi/
2025-10-18 22:03
-
controllercli/
2022-05-16 16:17
-
quickstart/
2025-10-18 22:03
-
release-notes/
2025-10-18 22:03
-
towerapi/
2025-10-18 22:03
-
towercli/
2022-05-16 16:17
-
upgrade-migration-gu..>
2025-10-18 22:03
-
userguide/
2025-10-18 22:03
-

Content from: https://docs.ansible.com/automation-controller/4.1.3/html/

Index of /automation-controller/4.1.3/html
Index of /automation-controller/4.1.3/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:13
-
controllerapi/
2025-10-18 22:13
-
quickstart/
2025-10-18 22:13
-
release-notes/
2025-10-18 22:13
-
towerapi/
2025-10-18 22:13
-
upgrade-migration-gu..>
2025-10-18 22:13
-
userguide/
2025-10-18 22:13
-

Content from: https://docs.ansible.com/automation-controller/4.2.0/

Index of /automation-controller/4.2.0
Index of /automation-controller/4.2.0
Name
Last modified
Size
Description
Parent Directory
-
access/
2022-09-01 02:11
-
html/
2025-10-18 22:39
-
html_ja/
2022-05-26 06:24
-
html_ko/
2022-05-26 06:10
-
html_zh/
2022-05-26 06:39
-
pdf/
2023-02-10 05:34
-

Content from: https://docs.ansible.com/automation-controller/4.2.0/html/

Index of /automation-controller/4.2.0/html
Index of /automation-controller/4.2.0/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:39
-
controllerapi/
2025-10-18 22:39
-
controllercli/
2022-06-02 18:55
-
quickstart/
2025-10-18 22:39
-
release-notes/
2025-10-18 22:39
-
towerapi/
2025-10-18 22:39
-
towercli/
2022-06-02 18:55
-
upgrade-migration-gu..>
2025-10-18 22:39
-
userguide/
2025-10-18 22:39
-

Content from: https://docs.ansible.com/automation-controller/4.2.1/

Index of /automation-controller/4.2.1
Index of /automation-controller/4.2.1
Name
Last modified
Size
Description
Parent Directory
-
html/
2025-10-18 22:52
-
html_ja/
2022-12-07 19:28
-
html_ko/
2022-12-07 17:58
-
html_zh/
2022-12-07 18:43
-
pdf/
2023-02-11 03:08
-

Content from: https://docs.ansible.com/automation-controller/4.2.1/html/

Index of /automation-controller/4.2.1/html
Index of /automation-controller/4.2.1/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:52
-
controllerapi/
2025-10-18 22:52
-
controllercli/
2022-11-30 19:10
-
quickstart/
2025-10-18 22:52
-
release-notes/
2025-10-18 22:52
-
towerapi/
2025-10-18 22:52
-
towercli/
2022-11-30 19:10
-
upgrade-migration-gu..>
2025-10-18 22:52
-
userguide/
2025-10-18 22:52
-

Content from: https://docs.ansible.com/automation-controller/4.2.2/html/

Index of /automation-controller/4.2.2/html
Index of /automation-controller/4.2.2/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:19
-
controllerapi/
2025-10-18 22:19
-
quickstart/
2025-10-18 22:19
-
release-notes/
2025-10-18 22:19
-
towerapi/
2025-10-18 22:19
-
upgrade-migration-gu..>
2025-10-18 22:19
-
userguide/
2025-10-18 22:19
-

Content from: https://docs.ansible.com/automation-controller/4.4/html/

Index of /automation-controller/4.4/html
Index of /automation-controller/4.4/html
Name
Last modified
Size
Description
Parent Directory
-
administration/
2025-10-18 22:49
-
controllerapi/
2025-10-18 22:49
-
controllercli/
2023-07-13 20:43
-
quickstart/
2025-10-18 22:49
-
release-notes/
2025-10-18 22:49
-
towerapi/
2025-10-18 22:49
-
towercli/
2023-07-13 20:43
-
upgrade-migration-gu..>
2025-10-18 22:49
-
userguide/
2025-10-18 22:49
-

Content from: https://docs.ansible.com/automation-controller/latest/html/controllercli/index.html

awx bulk
awx bulk get
awx bulk host_create
awx bulk host_delete
awx bulk job_launch
awx bulk list
awx config
awx constructed_inventory
awx constructed_inventory create
awx constructed_inventory delete
awx constructed_inventory get
awx constructed_inventory list
awx constructed_inventory modify
awx credential_input_sources
awx credential_input_sources create
awx credential_input_sources delete
awx credential_input_sources get
awx credential_input_sources list
awx credential_input_sources modify
awx credential_types
awx credential_types create
awx credential_types delete
awx credential_types get
awx credential_types list
awx credential_types modify
awx credentials
awx credentials create
awx credentials delete
awx credentials get
awx credentials list
awx credentials modify
awx execution_environments
awx execution_environments create
awx execution_environments delete
awx execution_environments get
awx execution_environments list
awx execution_environments modify
awx export
awx groups
awx groups create
awx groups delete
awx groups get
awx groups list
awx groups modify
awx host_metric_summary_monthly
awx host_metric_summary_monthly get
awx host_metric_summary_monthly list
awx host_metrics
awx host_metrics delete
awx host_metrics get
awx host_metrics list
awx hosts
awx hosts create
awx hosts delete

awx hosts get
awx hosts list
awx hosts modify
awx import
awx instance_groups
awx instance_groups create
awx instance_groups delete
awx instance_groups get
awx instance_groups list
awx instance_groups modify
awx instances
awx instances create
awx instances get
awx instances list
awx instances modify
awx inventory
awx inventory create
awx inventory delete
awx inventory get
awx inventory list
awx inventory modify
awx inventory_sources
awx inventory_sources associate
awx inventory_sources create
awx inventory_sources delete
awx inventory_sources disassociate
awx inventory_sources get
awx inventory_sources list
awx inventory_sources modify
awx inventory_sources update
awx inventory_updates
awx inventory_updates delete
awx inventory_updates get
awx inventory_updates list
awx inventory_updates stdout
awx job_templates
awx job_templates associate
awx job_templates create
awx job_templates delete
awx job_templates disassociate
awx job_templates get
awx job_templates launch
awx job_templates list
awx job_templates modify
awx jobs
awx jobs delete
awx jobs get
awx jobs list
awx jobs monitor
awx jobs stdout
awx labels
awx labels create
awx labels get
awx labels list

awx labels modify
awx login
awx me
awx mesh_visualizer
awx metrics
awx notification_templates
awx notification_templates create
awx notification_templates delete
awx notification_templates get
awx notification_templates list
awx notification_templates modify
awx notifications
awx notifications get
awx notifications list
awx organizations
awx organizations associate
awx organizations create
awx organizations delete
awx organizations disassociate
awx organizations get
awx organizations list
awx organizations modify
awx ping
awx project_updates
awx project_updates delete
awx project_updates get
awx project_updates list
awx project_updates stdout
awx projects
awx projects associate
awx projects create
awx projects delete
awx projects disassociate
awx projects get
awx projects list
awx projects modify
awx projects update
awx receptor_addresses
awx receptor_addresses get
awx receptor_addresses list
awx role_definitions
awx role_definitions create
awx role_definitions delete
awx role_definitions get
awx role_definitions list
awx role_definitions modify
awx role_team_assignments
awx role_team_assignments create
awx role_team_assignments delete
awx role_team_assignments get
awx role_team_assignments list
awx role_user_assignments
awx role_user_assignments create
awx role_user_assignments delete

awx role_user_assignments get
awx role_user_assignments list
awx roles
awx roles get
awx roles list
awx schedules
awx schedules create
awx schedules delete
awx schedules get
awx schedules list
awx schedules modify
awx service_index
awx service_index get
awx service_index list
awx settings
awx settings list
awx settings modify
awx system_job_templates
awx system_job_templates get
awx system_job_templates list
awx system_jobs
awx system_jobs delete
awx system_jobs get
awx system_jobs list
awx teams
awx teams create
awx teams delete
awx teams get
awx teams grant
awx teams list
awx teams modify
awx teams revoke
awx tokens
awx tokens create
awx tokens delete
awx tokens get
awx tokens list
awx tokens modify
awx unified_job_templates
awx unified_job_templates get
awx unified_job_templates list
awx unified_jobs
awx unified_jobs get
awx unified_jobs list
awx users
awx users create
awx users delete
awx users get
awx users grant
awx users list
awx users modify
awx users revoke
awx workflow_approvals
awx workflow_approvals delete

awx workflow_approvals get
awx workflow_approvals list
awx workflow_job_nodes
awx workflow_job_nodes get
awx workflow_job_nodes list
awx workflow_job_template_nodes
awx workflow_job_template_nodes create
awx workflow_job_template_nodes delete
awx workflow_job_template_nodes get
awx workflow_job_template_nodes list
awx workflow_job_template_nodes modify
awx workflow_job_templates
awx workflow_job_templates associate
awx workflow_job_templates create
awx workflow_job_templates delete
awx workflow_job_templates disassociate
awx workflow_job_templates get
awx workflow_job_templates launch
awx workflow_job_templates list
awx workflow_job_templates modify
awx workflow_jobs
awx workflow_jobs delete
awx workflow_jobs get
awx workflow_jobs list
awx workflow_jobs monitor
AnsibleFest
Products
Community
Webinars & Training
Blog
AWX CLI
Ansible Docs Home
»
Ansible Automation Platform Docs
»
? AWX CLI Automation Controller 4.6 documentation
»
AWX Command Line Interface
¶
awx is the official command-line client for AWX and Red Hat Ansible Automation Platform. It:
Uses naming and structure consistent with the AWX HTTP API
Provides consistent output formats with optional machine-parsable formats
To the extent possible, auto-detects API versions, available endpoints, and
feature support across multiple versions of AWX and Red Hat Ansible Automation Platform.
Potential uses include:
Configuring and launching jobs/playbooks
Checking on the status and output of job runs
Managing objects like organizations, users, teams, etc?
Basic Usage
Installation
Synopsis
Getting Started
Resources and Actions
Global Options

awx credential_input_sources modify
awx credential_types
awx credential_types create
awx credential_types delete
awx credential_types get
awx credential_types list
awx credential_types modify
awx credentials
awx credentials create
awx credentials delete
awx credentials get
awx credentials list
awx credentials modify
awx execution_environments
awx execution_environments create
awx execution_environments delete
awx execution_environments get
awx execution_environments list
awx execution_environments modify
awx export
awx groups
awx groups create
awx groups delete
awx groups get
awx groups list
awx groups modify
awx host_metric_summary_monthly
awx host_metric_summary_monthly get
awx host_metric_summary_monthly list
awx host_metrics
awx host_metrics delete
awx host_metrics get
awx host_metrics list
awx hosts
awx hosts create
awx hosts delete
awx hosts get
awx hosts list
awx hosts modify
awx import
awx instance_groups
awx instance_groups create
awx instance_groups delete
awx instance_groups get
awx instance_groups list
awx instance_groups modify
awx instances
awx instances create
awx instances get
awx instances list
awx instances modify
awx inventory
awx inventory create
awx inventory delete

awx inventory get
awx inventory list
awx inventory modify
awx inventory_sources
awx inventory_sources associate
awx inventory_sources create
awx inventory_sources delete
awx inventory_sources disassociate
awx inventory_sources get
awx inventory_sources list
awx inventory_sources modify
awx inventory_sources update
awx inventory_updates
awx inventory_updates delete
awx inventory_updates get
awx inventory_updates list
awx inventory_updates stdout
awx job_templates
awx job_templates associate
awx job_templates create
awx job_templates delete
awx job_templates disassociate
awx job_templates get
awx job_templates launch
awx job_templates list
awx job_templates modify
awx jobs
awx jobs delete
awx jobs get
awx jobs list
awx jobs monitor
awx jobs stdout
awx labels
awx labels create
awx labels get
awx labels list
awx labels modify
awx login
awx me
awx mesh_visualizer
awx metrics
awx notification_templates
awx notification_templates create
awx notification_templates delete
awx notification_templates get
awx notification_templates list
awx notification_templates modify
awx notifications
awx notifications get
awx notifications list
awx organizations
awx organizations associate
awx organizations create
awx organizations delete

awx organizations disassociate
awx organizations get
awx organizations list
awx organizations modify
awx ping
awx project_updates
awx project_updates delete
awx project_updates get
awx project_updates list
awx project_updates stdout
awx projects
awx projects associate
awx projects create
awx projects delete
awx projects disassociate
awx projects get
awx projects list
awx projects modify
awx projects update
awx receptor_addresses
awx receptor_addresses get
awx receptor_addresses list
awx role_definitions
awx role_definitions create
awx role_definitions delete
awx role_definitions get
awx role_definitions list
awx role_definitions modify
awx role_team_assignments
awx role_team_assignments create
awx role_team_assignments delete
awx role_team_assignments get
awx role_team_assignments list
awx role_user_assignments
awx role_user_assignments create
awx role_user_assignments delete
awx role_user_assignments get
awx role_user_assignments list
awx roles
awx roles get
awx roles list
awx schedules
awx schedules create
awx schedules delete
awx schedules get
awx schedules list
awx schedules modify
awx service_index
awx service_index get
awx service_index list
awx settings
awx settings list
awx settings modify
awx system_job_templates

awx system_job_templates get
awx system_job_templates list
awx system_jobs
awx system_jobs delete
awx system_jobs get
awx system_jobs list
awx teams
awx teams create
awx teams delete
awx teams get
awx teams grant
awx teams list
awx teams modify
awx teams revoke
awx tokens
awx tokens create
awx tokens delete
awx tokens get
awx tokens list
awx tokens modify
awx unified_job_templates
awx unified_job_templates get
awx unified_job_templates list
awx unified_jobs
awx unified_jobs get
awx unified_jobs list
awx users
awx users create
awx users delete
awx users get
awx users grant
awx users list
awx users modify
awx users revoke
awx workflow_approvals
awx workflow_approvals delete
awx workflow_approvals get
awx workflow_approvals list
awx workflow_job_nodes
awx workflow_job_nodes get
awx workflow_job_nodes list
awx workflow_job_template_nodes
awx workflow_job_template_nodes create
awx workflow_job_template_nodes delete
awx workflow_job_template_nodes get
awx workflow_job_template_nodes list
awx workflow_job_template_nodes modify
awx workflow_job_templates
awx workflow_job_templates associate
awx workflow_job_templates create
awx workflow_job_templates delete
awx workflow_job_templates disassociate
awx workflow_job_templates get
awx workflow_job_templates launch

awx workflow_job_templates list
awx workflow_job_templates modify
awx workflow_jobs
awx workflow_jobs delete
awx workflow_jobs get
awx workflow_jobs list
awx workflow_jobs monitor
Indices and tables
¶
Index
Module Index
Search Page
Next

Content from: https://docs.ansible.com/automation-controller/latest/html/release-notes/index.html

Automation Controller Release Notes v4.6 ? Automation Controller Release Notes v4.6
AnsibleFest
Products
Community
Webinars & Training
Blog
Documentation
AUTOMATION
CONTROLLER
v4.6
Are you using the latest and greatest version of Automation Controller? Find the
Automation Controller documentation
set which best matches your version of the controller.
Automation Controller Release Notes v4.6
Release Notes
Known Issues
The
CSRF_TRUSTED_ORIGIN
setting
Launching the ansible-runner component not working as expected
Deleted default orgs produces duplicate Ansible-Galaxy credentials
Isolated nodes unsupported in an OpenShift deployment
Browsers ignoring the
autocomplete=off
setting
Login via HTTP requires workaround
Job slicing and limit interactions
Misuse of job slicing can cause errors in job scheduling
Default LDAP directory must be configured to use LDAP Authentication
Potential security issue using
X_FORWARDED_FOR
in
REMOTE_HOST_HEADERS
Server error when accessing SAML metadata via hostname
SAML authentication revokes admin role upon login
Live events status indicators
VMWare Self-Signed Certs
awx-manage inventory_import user
Database on Disk Becomes Corrupted
Safari unable to establish connection to web socket
Local management not functioning as expected
Problems when using SSH customization
Database server installed on nodes
Reactivating OAuth authentication accounts which have been deleted
Using vaulted variables in inventory sourced from a project
Saved scheduled and workflow configurations and surveys
Supported Locales
AnsibleFest
Products
Community
Webinars & Training
Blog

Automation Controller Release Notes
Ansible Docs Home
»
Ansible Automation Platform Docs
»
? Automation Controller Release Notes v4.6
»
Automation Controller Release Notes v4.6
¶
Thank you for your interest in Red Hat Ansible Automation Platform controller. Automation controller is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.
The
Automation Controller Release Notes
provides release notes, known issues, and related reference materials. This document has been updated to include information for the latest release of Automation Controller v4.6.
We Need Feedback!
If you spot a typo in this documentation, or if you have thought of a way to make this manual better, we would love to hear from you! Please send an email to:
docs
@
ansible
.
com
If you have a suggestion, try to be as specific as possible when describing it. If you have found an error, please include the manual?s title, chapter number/section number, and some of the surrounding text so we can find it easily. We may not be able to respond to every message sent to us, but you can be sure that we will be reading them all!
Automation Controller Version 4.6; September 30, 2024;
https://access.redhat.com/
Automation Controller Release Notes v4.6
Release Notes
Known Issues
The
CSRF_TRUSTED_ORIGIN
setting
Launching the ansible-runner component not working as expected
Deleted default orgs produces duplicate Ansible-Galaxy credentials
Isolated nodes unsupported in an OpenShift deployment
Browsers ignoring the
autocomplete=off
setting
Login via HTTP requires workaround
Job slicing and limit interactions
Misuse of job slicing can cause errors in job scheduling
Default LDAP directory must be configured to use LDAP Authentication
Potential security issue using
X_FORWARDED_FOR
in
REMOTE_HOST_HEADERS
Server error when accessing SAML metadata via hostname
SAML authentication revokes admin role upon login
Live events status indicators
VMWare Self-Signed Certs
awx-manage inventory_import user

Database on Disk Becomes Corrupted
Safari unable to establish connection to web socket
Local management not functioning as expected
Problems when using SSH customization
Database server installed on nodes
Reactivating OAuth authentication accounts which have been deleted
Using vaulted variables in inventory sourced from a project
Saved scheduled and workflow configurations and surveys
Supported Locales
Index
¶
Index

Next

Red Hat Ansible Automation Platform docs are generated using
Sphinx
using a theme provided by
Read the Docs
.

Content from: https://docs.ansible.com/automation-tower-prior-versions.html

Automation Controller documentation archive | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Automation Controller documentation archive
Archive page for the Automation Controller documentation that provides older versions of content.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Automation Controller version 4.5 (Supported)
Automation Controller version 4.4 (Supported)
Automation Controller version 4.3 (Supported)
Automation Controller version 4.2.2 (Supported)
Automation Controller version 4.2.1 (Supported)
Automation Controller version 4.2.0 (Supported)
Automation Controller version 4.1.4 (Supported)
Automation Controller version 4.1.3 (Supported)
Automation Controller version 4.1.2 (Supported)
Automation Controller version 4.1.1 (Supported)
Automation Controller version 4.1.0 (Supported)
Automation Tower version 3.8.6 (Supported)
Automation Tower version 3.8.5 (Supported)
Automation Tower version 3.8.4 (Supported)
Automation Tower version 3.8.3 (Supported)
Automation Tower version 3.8.2 (Supported)
Automation Tower version 3.8.1 (Supported)
Automation Tower version 3.8.0 (Supported)

Content from: https://docs.ansible.com/community.html

Join the Ansible community | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Join the Ansible community
Open-source and collaboration are at the heart of the Ansible community, helping more people experience the power of automation and work better and faster together.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Collaborate with us
Read the code of conduct
Explore ways to contribute
Find an Ansible project
Review the contributor path
Join the conversation
Join the Ansible Forum
Find public community meetings on the Forum Events calendar
Check out the guide to community communication
Participate in community discussions
Share your ideas and vote in the Community topics
Take part in a working group
Follow Ansible community news and announcements
Subscribe to the Bullhorn

Sponsored by

Content from: https://docs.ansible.com/core-translated-ja.html

????????? (Japanese) | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
????????? (Japanese)
Access Japanese translations for Ansible Core documentation.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
ansible-core ?????? 2.15
Visit the documentation
ansible-core ?????? 2.14
Visit the documentation
ansible-core ?????? 2.13
Visit the documentation
ansible-core ?????? 2.12
Visit the documentation
ansible-core ?????? 2.11
Visit the documentation
Ansible ?????? 2.9
Visit the documentation

Content from: https://docs.ansible.com/core.html

Ansible Core | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible Core
Core includes the Ansible language and runtime, a set of built-in modules and command-line tools, and a framework for extending automation with collections.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Ansible Core
Ansible Core is the language and runtime that powers automation.
Ansible Core documentation
Ansible Test
Ansible Test is a command-line tool for performing sanity, unit, and integration testing with Ansible Core and collections.
Ansible Test documentation
????????? (Japanese)
Access Japanese translations for Ansible Core documentation.
Learn more
Ansible community documentation archive
Archive page for the Ansible community documentation that provides older versions of content.
Visit the archive

Content from: https://docs.ansible.com/dev_guide.html

Developer Guide ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Adding modules and plugins locally
Should you develop a module?
Developing modules
Contributing your module to an existing Ansible collection
Conventions, tips, and pitfalls
Ansible and Python 3
Debugging modules
Module format and documentation
Ansible markup
Adjacent YAML documentation files
Windows module development walkthrough
Creating a new collection
Testing Ansible
The lifecycle of an Ansible module or plugin
Developing plugins
Developing dynamic inventory
Developing
ansible-core

Developer Guide
?
Note
Making Open Source More Inclusive
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see
our CTO Chris Wright?s message
.
Welcome to the Ansible Developer Guide!

Who should use this guide?

If you want to extend Ansible by using a custom module or plugin locally, creating a module or plugin, adding functionality to an existing module, or expanding test coverage, this guide is for you. We?ve included detailed information for developers on how to test and document modules, as well as the prerequisites for getting your module or plugin accepted into the main Ansible repository.

Find the task that best describes what you want to do:

I?m looking for a way to address a use case:

I want to
add a custom plugin or module locally
.

I want to figure out if
developing a module is the right approach
for my use case.

I want to understand
what a successful collection creator path looks like
.

I want to
develop a collection
.

I want to
contribute to an Ansible-maintained collection
.

I want to
contribute to a community-maintained collection
.

I want to
migrate a role to a collection
.

I?ve read the info above, and I?m sure I want to develop a module:

What do I need to know before I start coding?

I want to
set up my Python development environment
.

I want to
get started writing a module
.

I want to write a specific kind of module:

a
network module

a
Windows module
.

an
Amazon module
.

an
oVirt/RHV module
.

a
VMware module
.

I want to
write a series of related modules
that integrate Ansible with a new product (for example, a database, cloud provider, network platform, and so on).

I want to refine my code:
I want to
debug my module code
.
I want to
add tests
.
I want to
document my module
.
I want to
improve documentation by using Ansible markup
.
I want to
document my set of modules for a network platform
.
I want to follow
conventions and tips for clean, usable module code
.
I want to
make sure my code runs on Python 2 and Python 3
.
I want to work on other development projects:
I want to
write a plugin
.
I want to
connect Ansible to a new source of inventory
.
I want to
deprecate an outdated module
.
I want to contribute back to the Ansible project:
I want to
understand how to contribute to Ansible
.
I want to
contribute my module or plugin
.
I want to
understand the DCO agreement
for contributions to the
Ansible Core
and
Ansible Documentation
repositories.
If you prefer to read the entire guide, here?s a list of the pages in order.
Adding modules and plugins locally
Modules and plugins: what is the difference?
Adding modules and plugins in collections
Adding a module or plugin outside of a collection
Adding a non-module plugin locally outside of a collection
Using
ansible.legacy

Windows environment setup
Create a Windows server in a VM
Create an Ansible inventory
Provisioning the environment
Windows new module development
Windows module utilities
Windows playbook module testing
Windows debugging
Windows unit testing
Windows integration testing
Windows communication and development support
Creating a new collection
Before you start coding
Naming conventions
Speak to us
Where to get support
Required files
New to Git or GitHub
Testing Ansible
Why test your Ansible contributions?
Types of tests
Testing within GitHub & Azure Pipelines
How to test a PR
Want to know more about testing?
The lifecycle of an Ansible module or plugin
Deprecating modules and plugins in the Ansible main repository
Deprecating modules and plugins in a collection
Changing a module or plugin name in the Ansible main repository
Renaming a module or plugin in a collection, or redirecting a module or plugin to another collection
Tombstoning a module or plugin in a collection
Developing plugins
Writing plugins in Python
Raising errors
String encoding
Plugin configuration & documentation standards
Developing particular plugin types
Developing dynamic inventory
Inventory sources
Inventory plugins
Inventory scripts
Developing
ansible-core
ansible-core
project branches and tags
Ansible module architecture
Ansible module architecture
Types of modules
How modules are executed
Python API
Rebasing a pull request
Configuring your remotes
Rebasing your branch
Updating your pull request
Getting help rebasing

Using and developing module utilities
Naming and finding module utilities
Standard module utilities
Ansible collection creator path
Examine currently available solutions
Create your content
Put your content in a collection
Write good user collection documentation
Publish your collection source code
Follow a versioning convention
Understand and implement testing and CI
Provide good contributor & maintainer documentation
Publish your collection on distribution servers
Make your collection a part of Ansible community package
Maintain
Communicate
Developing collections
Creating collections
Using shared resources in collections
Testing collections
Distributing collections
Documenting collections
Migrating Ansible content to a different collection
Contributing to collections
Generating changelogs and porting guide entries in a collection
Collection structure
Collection Galaxy metadata structure
Migrating Roles to Roles in Collections on Galaxy
Comparing standalone roles to collection roles
Migrating a role to a collection
Migrating a role that contains plugins to a collection
Using
ansible.legacy
to access local custom modules from collections-based roles
Collection Galaxy metadata structure
Structure
Examples
Ansible architecture
Modules
Module utilities
Plugins
Inventory
Playbooks
The Ansible search path
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developers.html

Developers | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Developers
Extend automation with custom Ansible modules, add functionality to existing modules, or fix bugs to improve existing code.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
Developing modules
Ansible and Python 3
Python API
Start writing code
Set up your development environment
Learn how Ansible works
Write custom modules or plugins
Contribute code to a collection
Make your first contribution
Explore the Collection contributor guide
Contribute your module to an existing collection
Review the lifecycle of an Ansible module or plugin
Test plugins and modules
Understand testing in Ansible
Run sanity tests
Write integration tests
Write unit tests
Create new collections
Set things up with the collection skeleton
Test your collection for code quality
Publish your collection on a distribution server
Request a collection be added to the Ansible package

Content from: https://docs.ansible.com/developing_api.html

Python API ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Python API

Attention

The Ansible API is intended for internal Ansible use. Ansible may make changes to this API at any time that could break backward compatibility with older versions of the API. Because of this, external use is not supported by Ansible. If you want to use Python API only for executing playbooks or modules, consider
ansible-runner
first.

If you would like to use Ansible programmatically from a language other than Python, trigger events asynchronously, or have access control and logging demands, please see the
AWX project
.

See also

Developing dynamic inventory
Developing dynamic inventory integrations
Developing modules
Getting started on developing a module
Developing plugins
How to develop plugins
Communication

Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide

Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections.html

Developing collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Developing collections

?

Collections are a distribution format for Ansible content. You can package and distribute playbooks, roles, modules, and plugins using collections. A typical collection addresses a set of related use cases. For example, the
cisco.ios
collection automates management of Cisco IOS devices.

You can create a collection and publish it to
Ansible Galaxy
or to a private Automation Hub instance. You can publish certified collections to the Red Hat Automation Hub, part of the Red Hat Ansible Automation Platform.

Examine the
Ansible collection creator path
to understand how to go from creating a collection to having it included in the Ansible package distribution.

Migrating Ansible content to a different collection
Migrating content
Contributing to collections
Contributing to a collection: community.general
Generating changelogs and porting guide entries in a collection
Understanding antsibull-changelog
Including collection changelogs into Ansible
Collections references
Collection structure
Collection directories and files
Collection Galaxy metadata structure
Structure
Examples
For instructions on developing modules, see
Developing modules
.
See also
Using Ansible collections
Learn how to install and use collections in playbooks and roles
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_changelogs.html

Generating changelogs and porting guide entries in a collection ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Generating changelogs and porting guide entries in a collection

?

You can create and share changelog and porting guide entries for your collection. If your collection is part of the Ansible Community package, we recommend that you use the
antsibull-changelog
tool to generate Ansible-compatible changelogs. The Ansible changelog uses the output of this tool to collate all the collections included in an Ansible release into one combined changelog for the release.

Note

Ansible here refers to the Ansible 2.10 or later release that includes a curated set of collections.

Understanding antsibull-changelog

?

The
antsibull-changelog
tool allows you to create and update changelogs for Ansible collections that are compatible with the combined Ansible changelogs. This is an update to the changelog generator used in prior Ansible releases. The tool adds three new changelog fragment categories:
breaking_changes
,
security_fixes
and
trivial
. The tool also generates the
changelog.yaml
file that Ansible uses to create the combined
CHANGELOG.rst
file and Porting Guide for the release.

See
Creating a changelog fragment
and the
antsibull-changelog documentation
for complete details.

Note
The collection maintainers set the changelog policy for their collections. See the individual collection contributing guidelines for complete details.
Generating changelogs
?
To initialize changelog generation:
Install
antsibull-changelog
:
pip
install
antsibull-changelog
.
Initialize changelogs for your repository:
antsibull-changelog
init
<path/to/your/collection>
.
Optionally, edit the
changelogs/config.yaml
file to customize the location of the generated changelog
.rst
file or other options. See
Bootstrapping changelogs for collections
for details.
To generate changelogs from the changelog fragments you created:
Optionally, validate your changelog fragments:
antsibull-changelog
lint
.
Generate the changelog for your release:
antsibull-changelog
release
[--version
version_number]
.
Note
Add the
--reload-plugins
option if you ran the
antsibull-changelog
release
command previously and the version of the collection has not changed.
antsibull-changelog
caches the information on all plugins and does not update its cache until the collection version changes.
Porting Guide entries from changelog fragments
?
The Ansible changelog generator automatically adds several changelog fragment categories to the Ansible Porting Guide:
major_changes
breaking_changes
deprecated_features
removed_features
Including collection changelogs into Ansible

?
If your collection is part of Ansible, use one of the following three options to include your changelog into the Ansible release changelog:
Use the
antsibull-changelog
tool.
If are not using this tool, include the properly formatted
changelog.yaml
file into your collection. See the
changelog.yaml format
for details.
Add a link to own changelogs or release notes in any format by opening an issue at
https://github.com/ansible-community/ansible-build-data/
with the HTML link to that information.
Note
For the first two options, Ansible pulls the changelog details from Galaxy so your changelogs must be included in the collection version on Galaxy that is included in the upcoming Ansible release.
See also
Generating changelogs and porting guide entries in a collection
Learn how to create good changelog fragments.
Using Ansible collections
Learn how to install and use collections.
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_contributing.html

Contributing to collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing collections
Contributing to collections
Edit on GitHub
Contributing to collections
?
If you want to add functionality to an existing collection, modify a collection you are using to fix a bug, or change the behavior of a module in a collection, clone the Git repository for that collection and make changes on a branch. You can combine changes to a collection with a local checkout of Ansible (
source
hacking/env-setup
).
You should first check the collection repository to see if it has specific contribution guidelines. These are typically listed in the README.md or CONTRIBUTING.md files within the repository.
See
Creating your first collection pull request
for more general guidelines and
Testing Ansible and Collections
for testing guidelines.
Contributing to a collection: community.general
?
These instructions apply to collections hosted in the
ansible_collections GitHub organization
. For other collections, especially for collections not hosted on GitHub, check the
README.md
of the collection for information on contributing to it.
This example uses the
community.general collection
. To contribute to other collections in the same GitHub org, replace the folder names
community
and
general
with the namespace and collection name of a different collection.
Prerequisites
?
Include
~/dev/ansible/collections/
in
COLLECTIONS_PATHS

If that path mentions multiple directories, make sure that no other directory earlier in the search path contains a copy of community.general
.
Creating a PR
?
Create the directory
~/dev/ansible/collections/ansible_collections/community
:
mkdir
-p
~/dev/ansible/collections/ansible_collections/community
Clone
the community.general Git repository
or a fork of it into the directory
general
:
cd
~/dev/ansible/collections/ansible_collections/community
git
clone

:ansible-collections/community.general.git
general
If you clone from a fork, add the original repository as a remote
upstream
:
cd
~/dev/ansible/collections/ansible_collections/community/general
git
remote
add
upstream

:ansible-collections/community.general.git
Create a branch and commit your changes on the branch.
Remember to add tests for your changes, see
Testing collections
.
Push your changes to your fork of the collection and create a Pull Request.
You can test your changes by using this checkout of
community.general
in playbooks and roles with whichever version of Ansible you have installed locally, including a local checkout of ansible/ansible
?s
devel
branch.
See also
Using Ansible collections
Learn how to install and use collections.
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous

Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_creating.html

Creating collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing collections
Creating collections
Edit on GitHub
Creating collections
?
To create a collection:
Create a
new collection
, optionally using a custom
collection template
, with the
ansible-galaxy
collection
init
command.
Add modules and other content to the collection.
Build the collection into a collection artifact with
ansible-galaxy collection build
.
Publish the collection artifact to Galaxy with
ansible-galaxy collection publish
.
A user can then install your collection on their systems.
Naming your collection
Creating a new collection
Creating a collection from a custom template
Creating collections with ansible-creator
Naming your collection
?
Collection names consist of a namespace and a name, separated by a period (
.
). Both namespace and name should be valid Python identifiers. This means that they should consist of ASCII letters, digits, and underscores.
Note
Usually namespaces and names use lower-case letters, digits, and underscores, but no upper-case letters.
You should make sure that the namespace you use is not registered by someone else by checking on
Ansible Galaxy?s namespace list
. If you chose a namespace or even a full collection name that collides with another collection on Galaxy, it can happen

that if you or someone else runs
ansible-galaxy
collection
install
with your collection name, you end up with another collection. Even if the namespace currently does not exist, it could be created later by someone else.
If you want to request a new namespace on Ansible Galaxy,
create an issue on github.com/ansible/galaxy
.
There are a few special namespaces:
ansible
:
The
ansible namespace
is owned by Red Hat and reserved for official Ansible collections. Two special members are the synthetic
ansible.builtin
and
ansible.legacy
collections. These cannot be found on Ansible Galaxy, but are built-in into ansible-core.
community
:
The
community namespace
is owned by the Ansible community. Collections from this namespace generally live in the
GitHub ansible-collection organization
. If you want to create a collection in this namespace,
request
it on the forum.
local
:
The
local namespace
does not contain any collection on Ansible Galaxy, and the intention is that this will never change. You can use the
local
namespace for collections that are locally on your machine or locally in your Git repositories, without having to fear collisions with actually existing collections on Ansible Galaxy.
Creating a new collection
?
Create your collection skeleton in a path that includes
ansible_collections
, for example
collections/ansible_collections/
.
To start a new collection, run the following command in your collections directory:
ansible_collections#>
ansible-galaxy
collection
init
my_namespace.my_collection
Note
Both the namespace and collection names use the same strict set of requirements. Both are limited to alphanumeric characters and underscores, must have a minimum length of two characters, and cannot start with an underscore.
It will create the structure
[my_namespace]/[my_collection]/[collection

skeleton]
.
Hint
If Git is used for version control, the corresponding repository should be initialized in the collection directory.
Once the collection exists, you can populate the directories with the content you want inside the collection. See
ansible-collections
GitHub Org to get a better idea of what you can place inside a collection.
Reference: the
ansible-galaxy
collection
command
Currently the
ansible-galaxy
collection
command implements the following sub commands:
init
: Create a basic collection based on the default template included with Ansible or your own template.
build
: Create a collection artifact that can be uploaded to Galaxy or your own repository.
publish
: Publish a built collection artifact to Galaxy.
install
: Install one or more collections.
To learn more about the
ansible-galaxy
command-line tool, see the
ansible-galaxy
man page.
Creating a collection from a custom template
?
The built-in collection template is a simple example of a collection that works with
ansible-core
, but if you want to simplify your development process you may want to create a custom collection template to pass to
ansible-galaxy
collection
init
.
A collection skeleton is a directory that looks like a collection directory but any
.j2
files (excluding those in
templates/
and
roles/*/templates/
) will be templated by
ansible-galaxy
collection
init
. The skeleton?s
galaxy.yml.j2
file should use the variables
namespace
and
collection_name
which are derived from

ansible-galaxy
init
namespace.collection_name
, and will populate the metadata in the initialized collection?s
galaxy.yml
file. There are a few additional variables available by default (for example,
version
is
1.0.0
), and these can be supplemented/overridden using
--extra-vars
.
An example
galaxy.yml.j2
file that accepts an optional dictionary variable
dependencies
could look like this:

```
namespace:
{{
namespace
}}
name:
{{
collection_name
}}
version:
{{
(
version
|
quote
)
is
version
(
'0.0.0'
,
operator
=
'gt'
,
version_type
=
'semver'
)|
ternary
(
version
,
undef
(
'version must be a valid semantic version greater than 0.0.0'
))
}}
```

dependencies:
{{
dependencies
|
default
({},
true
)
}}
To initialize a collection using the new template, pass the path to the skeleton with
ansible-galaxy
collection
init
:
ansible_collections#>
ansible-galaxy
collection
init
--collection-skeleton
/path/to/my/namespace/skeleton
--extra-vars
"@my_vars_file.json"
my_namespace.my_collection
Note
Before
ansible-core
2.17, collection skeleton templating is limited to the few hardcoded variables including
namespace
,
collection_name
, and
version
.
Note
The default collection skeleton uses an internal filter
comment_ify
that isn?t accessibly to
--collection-skeleton
. Use
ansible-doc
-t
filter|test
--list
to see available plugins.
Creating collections with ansible-creator
?
ansible-creator
is designed to quickly scaffold an Ansible collection project.
Note
The
Ansible Development Tools
package offers a convenient way to install
ansible-creator
along with a curated set of tools for developing automation content.

After
installing
ansible-creator
you can initialize a project in one of the following ways:
Use the
init
subcommand.
Use
ansible-creator
with the
Ansible extension
in Visual Studio Code.
See also
Using Ansible collections
Learn how to install and use collections.
Collection structure
Directories and files included in the collection skeleton
Ansible Development Tools (ADT)
Python package of tools to create and test Ansible content.
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_documenting.html

Documenting collections ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing collections
Documenting collections
Edit on GitHub
Documenting collections
?
Documenting modules and plugins
?
Documenting modules is thoroughly documented in
Module format and documentation
. Plugins can be documented the same way as modules, that is with
DOCUMENTATION
,
EXAMPLES
, and
RETURN
blocks.
Documenting roles
?
To document a role, you have to add a role argument spec by creating a file
meta/argument_specs.yml
in your role. See
Role argument validation
for details. As an example, you can look at
the argument specs file
of the
telekom_mms.icinga_director.ansible_icinga role
on GitHub.
Verifying your collection documentation
?
You can use
antsibull-docs
to lint your collection documentation.
See
Linting collection documentation
.
for details.
Build a docsite with antsibull-docs
?

You can use
antsibull-docs
to build a Sphinx-based docsite for your collection:
Create your collection and make sure you can use it with ansible-core by adding it to your
COLLECTIONS_PATHS
.
Create a directory
dest
and run
antsibull-docs
sphinx-init
--use-current
--dest-dir
dest
namespace.name
, where
namespace.name
is the name of your collection.
Go into
dest
and run
pip
install
-r
requirements.txt
. You might want to create a venv and activate it first to avoid installing this globally.
Then run
./build.sh
.
Open
build/html/index.html
in a browser of your choice.
See
antsibull-docs documentation
for complete details.
If you want to add additional documentation to your collection next to the plugin, module, and role documentation, see
docs directory
.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_migrating.html

Migrating Ansible content to a different collection ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Migrating Ansible content to a different collection

?

You might decide to move content from one collection to another; for example, to extract a set of related modules out of
community.general
or
community.network
to create a more focused collection.

When you migrate content between collections, you must take certain steps to ensure users can follow the transition.

Migrating content

?

If the collection from which you are going to migrate content is included in the
Ansible community package
, ensure the target collection satisfies the
Ansible community package collections requirements
. After you satisfy the requirements, you can migrate the content as follows:

Copy content from the source (old) collection to the target (new) collection.

Change
M()
, examples,
seealso
,
extended_documentation_fragments
to use actual FQCNs in moved content, old collection, and in other collections that have references to the content.

Move all related issues, pull requests, and wiki pages.

Look through the
docs/docsite
directory of the
ansible-documentation GitHub repository
(for example, using the
grep
command-line utility) to check if there are examples using the moved modules and plugins so that you can update those

FQCNs.
Deprecate the module/plugin with
removal_version
scheduled for the next major version in
meta/runtime.yml
of the old collection. The deprecation must be released after the copied content has been included in a release of the new collection.
When the next major release of the old collection is prepared:
remove the module/plugin from the old collection
remove related unit and integration tests
remove specific module utils (if they are NOT used by other modules/plugins or
module_utils
)
remove specific documentation fragments if there are any in the old collection
add a changelog fragment containing entries for
removed_features
and
breaking_changes
; you can see an example of a changelog fragment in this
pull request
change
meta/runtime.yml
in the old collection:
add
redirect
to the corresponding module/plugin?s entry
in particular, add
redirect
for the removed module utils and documentation fragments if applicable
remove
removal_version
from there
remove related entries from
tests/sanity/ignore.txt
files if exist
remove changelog fragments for removed content that are not yet part of the changelog (in other words, do not modify
changelogs/changelog.yaml
and do not delete files mentioned in it)
remove requirements that are no longer required in
tests/unit/requirements.txt
,
tests/requirements.yml
and
galaxy.yml
To implement these changes, you need to create at least three PRs:
Create a PR against the new collection to copy the content.
Deprecate the module/plugin in the old collection.
Later create a PR against the old collection to remove the content according to the schedule.
Adding the content to the new collection
?
Create a PR in the new collection to:
Copy ALL the related files from the old collection.
If it is an action plugin, include the corresponding module with documentation.
If it is a module, check if it has a corresponding action plugin that should move with it.

Check
meta/
for relevant updates to
runtime.yml
if it exists.
Carefully check the moved
tests/integration
and
tests/units
and update for FQCN.
Review
tests/sanity/ignore-*.txt
entries in the old collection.
Update
meta/runtime.yml
in the old collection.
Removing the content from the old collection
?
Create a PR against the source collection repository to remove the modules, module_utils, plugins, and docs_fragments related to this migration:
If you are removing an action plugin, remove the corresponding module that contains the documentation.
If you are removing a module, remove any corresponding action plugin that should stay with it.
Remove any entries about removed plugins from
meta/runtime.yml
. Ensure they are added into the new repo.
Remove sanity ignore lines from
tests/sanity/ignore*.txt
Remove associated integration tests from
tests/integrations/targets/
and unit tests from
tests/units/plugins/
.
if you are removing from content from
community.general
or
community.network
, remove entries from
.github/BOTMETA.yml
.
Carefully review
meta/runtime.yml
for any entries you may need to remove or update, in particular deprecated entries.
Update
meta/runtime.yml
to contain redirects for EVERY PLUGIN, pointing to the new collection name.
Warning
Maintainers for the old collection have to make sure that the PR is merged in a way that it does not break user experience and semantic versioning:
A new version containing the merged PR must not be released before the collection the content has been moved to has been released again, with that content contained in it. Otherwise the redirects cannot work and users relying on that content will experience breakage.
Once 1.0.0 of the collection from which the content has been removed has been released, such PRs can only be merged for a new
major

version (in other words, 2.0.0, 3.0.0, and so on).
Updating BOTMETA.yml
?
The
BOTMETA.yml
, for example in
community.general collection repository
, is the source of truth for:
ansibullbot
If the old and/or new collection has
ansibullbot
, its
BOTMETA.yml
must be updated correspondingly.
Ansibulbot will know how to redirect existing issues and PRs to the new repo. The build process for docs.ansible.com will know where to find the module docs.
$modules/monitoring/grafana/grafana_plugin.py
:
migrated_to
:
community.grafana
$modules/monitoring/grafana/grafana_dashboard.py
:
migrated_to
:
community.grafana
$modules/monitoring/grafana/grafana_datasource.py
:
migrated_to
:
community.grafana
$plugins/callback/grafana_annotations.py
:
maintainers
:
$team_grafana
labels
:
monitoring grafana
migrated_to
:
community.grafana
$plugins/doc_fragments/grafana.py
:
maintainers
:
$team_grafana
labels
:
monitoring grafana
migrated_to
:
community.grafana
Example PR

The
migrated_to:
key must be added explicitly for every
file
. You cannot add
migrated_to
at the directory level. This is to allow module and plugin webdocs to be redirected to the new collection docs.
migrated_to:
MUST be added for every:
module
plugin
module_utils
contrib/inventory script
You do NOT need to add
migrated_to
for:
Unit tests
Integration tests
ReStructured Text docs (anything under
docs/docsite/rst/
)
Files that never existed in
ansible/ansible:devel
See also
Using Ansible collections
Learn how to install and use collections.
Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_collections_path.html

Ansible collection creator path ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Ansible collection creator path
?
Note
If you are unfamiliar with Ansible collections, first take a look at the
Using Ansible collections guide
.
Ansible collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.
A typical collection addresses a set of related use cases. For example, the
community.dns
collection includes modules and plugins to work with DNS.
You can install collections made by others or share yours with the community through a distribution server such as
Ansible Galaxy
. Certified collections can be published to the Red Hat Automation Hub, a part of the Red Hat Ansible Automation Platform.
Creating and sharing collections is a great way of contributing to the Ansible project.
The Ansible community package consists of
ansible-core
, which, among other core components, includes the
ansible.builtin
collection maintained by the Core team, and a set of collections maintained by the community.
The purpose of this guide is to give you as a (potential) content creator a consistent overview of the Ansible collection creator journey from an idea for the first module/role to having your collection included in the Ansible community package. The
Collection development guidelines section
provides references to more detailed aspects of this journey.
The overall journey consists of the following milestones:
Examine currently available solutions
Create your content
Put your content in a collection
Write good user collection documentation
Publish your collection source code
Follow a versioning convention
Understand and implement testing and CI
Add tests
Implement continuous integration
Provide good contributor & maintainer documentation

Publish your collection on distribution servers
Make your collection a part of Ansible community package
Maintain
Communicate
Examine currently available solutions
?
If you have an idea for a new role or module/plugin, there is no need to reinvent the wheel if there is already a sufficient solution that solves your automation issue.
Therefore, first examine the currently available content including:
Ansible builtin modules and plugins
Ansible package collection index
Ansible Galaxy
Ansible Automation Hub
if you have the Ansible Automation Platform subscription
In case the solutions you found are not fully sufficient or have flaws, consider improving them rather than creating your own. Each collection includes information on where to create issues for that collection to propose your enhancement ideas.
If you already have your content written and used in your workflows, you can still consider integrating it to the existing solutions.
However, if these options do not apply to your collection ideas, we encourage you to create and share your own.
Create your content
?
You
tried
but have not found any sufficient solution for your automation issue.
Use one of the following guides:
Roles guide
: if you want to create a role.
Developer guide
: if you want to create a new Ansible module or plugin for your personal use.
Put your content in a collection
?
You
created
new content.
Now it is time to create a reusable and sharable collection.
Use the
Developing collections guide
to learn how.
We recommend you to use the
collection_template repository
as a basis for your collection.
Write good user collection documentation
?
Your collection
README.md
file should contain a quick-start installation and usage guides.
You can use the
community.general collection README file
as an example.
If your collection contains modules or plugins, make sure their documentation is comprehensive.
Use the
Module format and documentation guide
and

Ansible documentation style guide
to learn more.
Publish your collection source code
?
Publish your collection on a platform for software development and version control such as
GitHub
.
It can be your personal repository or your organization?s one.
You can also
request
a repository under the
ansible-collections
organization.
Make sure your collection contains exhaustive license information.
Ansible is an open source project, so we encourage you to license it under one of open source licenses.
If you plan to submit your collection for inclusion in the Ansible community package, your collection must satisfy the licensing requirements
.
If you have used the
collection_template repository
we recommended earlier as a skeleton for your collection, it already contains the
GNU
GPL
v3
license.
Follow a versioning convention
?
When releasing new versions of your collections, take the following recommended practices into consideration:
Follow a versioning convention. Using
SemVer
is highly recommended.
Base your releases on
Git tags
.
Understand and implement testing and CI
?
This section is applicable to collections containing modules and plugins.
For role testing, see the
Ansible Molecule
project.
Add tests
?
Testing your collection ensures that your code works well and integrates with other components such as
ansible-core
.
Take a look at the following documents:
Testing Ansible guide
: provides general information about testing.
Testing collections guide
: contains collection-specific testing information.
Implement continuous integration
?
Now make sure when pull requests are created in your collection repository they are automatically tested using a CI tool such as GitHub Actions or Azure Pipelines.

The
collection_template repository
contains GitHub Actions
templates
you can adjust and use to enable the workflows in your repository.
Provide good contributor & maintainer documentation
?
See the
collection_template/README.md
as an example.
Publish your collection on distribution servers
?
To distribute your collection and allow others to conveniently use it, publish your collection on one or more distribution servers.
See the
Distributing collections guide
to learn how.
Make your collection a part of Ansible community package
?
Make you collection satisfy the
Ansible community package collections requirements
and submit it for inclusion.
See the
inclusion process description
to learn how.
Maintain
?
Maintain your collection.
See the
Ansible collection maintainer guidelines
for details.
Communicate
?
Engage with the community.
Take a look at the
Ansible communication guide
to see available communication options.
See also
Developing collections
A set of guidelines about collection development aspects
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_core.html

Developing ansible-core ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing
ansible-core
Edit on GitHub
Developing
ansible-core
?
Although
ansible-core
(the code hosted in the
ansible/ansible repository
on GitHub) includes a few plugins that can be swapped out by the playbook directives or configuration, much of the code there is not modular. The documents here give insight into how the parts of
ansible-core
work together.
ansible-core
project branches and tags
Ansible module architecture
See also
Python API
Learn about the Python API for task execution
Developing plugins
Learn about developing plugins
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_inventory.html

Developing dynamic inventory ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Developing dynamic inventory
?
Ansible can pull inventory information from dynamic sources, including cloud sources, by using the supplied
inventory plugins
. For details about how to pull inventory information, see
Working with dynamic inventory
. If the source you want is not currently covered by existing plugins, you can create your own inventory plugin as with any other plugin type.
In previous versions, you had to create a script or program that could output JSON in the correct format when invoked with the proper arguments.
You can still use and write inventory scripts, as we ensured backwards compatibility through the
script inventory plugin
and there is no restriction on the programming language used.
If you choose to write a script, however, you will need to implement some features yourself such as caching, configuration management, dynamic variable and group composition, and so on.
If you use
inventory plugins
instead, you can use the Ansible codebase and add these common features automatically.
Topics

Inventory sources
?
Inventory sources are the input strings that inventory plugins work with.
An inventory source can be a path to a file or to a script, or it can be raw data that the plugin can interpret.

The table below shows some examples of inventory plugins and the source types that you can pass to them with
-i
on the command line.
Plugin
Source
host list
A comma-separated list of hosts
yaml
Path to a YAML format data file
constructed
Path to a YAML configuration file
ini
Path to an INI formatted data file
virtualbox
Path to a YAML configuration file
script plugin
Path to an executable that outputs JSON
Inventory plugins
?
Like most plugin types (except modules), inventory plugins must be developed in Python. They execute on the control node and should therefore adhere to the
Control node requirements
.
Most of the documentation in
Developing plugins
also applies here. You should read that document first for a general understanding and then come back to this document for specifics on inventory plugins.
Normally, inventory plugins are executed at the start of a run, and before the playbooks, plays, or roles are loaded.
However, you can use the
meta:
refresh_inventory
task to clear the current inventory and execute the inventory plugins again, and this task will generate a new inventory.
If you use the persistent cache, inventory plugins can also use the configured cache plugin to store and retrieve data. Caching inventory avoids making repeated and costly external calls.
Developing an inventory plugin
?
The first thing you want to do is use the base class:
from
ansible.plugins.inventory
import
BaseInventoryPlugin
class
InventoryModule
(
BaseInventoryPlugin
):
NAME
=
'myplugin'
used internally by Ansible, it should match the file name but not required
If the inventory plugin is in a collection, the NAME should be in the ?namespace.collection_name.myplugin? format. The base class has a couple of methods that each plugin should implement and a few helpers for parsing the inventory source and updating the inventory.
After you have the basic plugin working, you can incorporate other features by adding more base classes:

```
from
ansible.plugins.inventory
import
BaseInventoryPlugin
,
Constructable
,
Cacheable
class
InventoryModule
(
BaseInventoryPlugin
,
Constructable
,
Cacheable
):
NAME
=
'myplugin'
```

For the bulk of the work in a plugin, we mostly want to deal with 2 methods `verify_file` and `parse`.

`verify_file` method?

Ansible uses this method to quickly determine if the inventory source is usable by the plugin. The determination does not need to be 100% accurate, as there might be an overlap in what plugins can handle and by default Ansible will try the enabled plugins as per their sequence.

```
def
verify_file
(
self
,
path
):
''' return true/false if this is possibly a valid file for this plugin to consume '''
valid
=
False
if
super
(
InventoryModule
,
self
)
.
verify_file
(
path
):
# base class verifies that file exists and is readable by current user
```

if
path
.
endswith
((
'virtualbox.yaml'
,
'virtualbox.yml'
,
'vbox.yaml'
,
'vbox.yml'
)):
valid
=
True
return
valid
In the above example, from the
virtualbox inventory plugin
, we screen for specific file name patterns to avoid attempting to consume any valid YAML file. You can add any type of condition here, but the most common one is ?extension matching?. If you implement extension matching for YAML configuration files, the path suffix <plugin_name>.<yml|yaml> should be accepted. All valid extensions should be documented in the plugin description.
The following is another example that does not use a ?file? but the inventory source string itself,
from the
host list
plugin:
def
verify_file
(
self
,
path
):
''' don't call base class as we don't expect a path, but a host list '''
host_list
=
path
valid
=
False
b_path
=
to_bytes
(
host_list
,
errors
=
'surrogate_or_strict'
)
if
not

os
.
path
.
exists
(
b_path
)
and
','
in
host_list
:
the path does NOT exist and there is a comma to indicate this is a 'host list'
valid
=
True
return
valid
This method is just to expedite the inventory process and avoid unnecessary parsing of sources that are easy to filter out before causing a parse error.
parse method
?
This method does the bulk of the work in the plugin.
It takes the following parameters:
inventory: inventory object with existing data and the methods to add hosts/groups/variables to inventory
loader: Ansible?s DataLoader. The DataLoader can read files, auto load JSON/YAML and decrypt vaulted data, and cache read files.
path: string with inventory source (this is usually a path, but is not required)
cache: indicates whether the plugin should use or avoid caches (cache plugin and/or loader)
The base class does some minimal assignment for reuse in other methods.
def
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):
self
.
loader
=
loader
self
.
inventory

=
inventory
self
.
templar
=
Templar
(
loader
=
loader
)
It is up to the plugin now to parse the provided inventory source and translate it into Ansible inventory.
To facilitate this, the example below uses a few helper functions:
NAME
=
'myplugin'
def
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):
call base method to ensure properties are available for use with other helper methods
super
(
InventoryModule
,
self
)
.
parse
(
inventory
,
loader
,
path
,
cache
)
this method will parse 'common format' inventory sources and
update any options declared in DOCUMENTATION as needed
config
=

```
self
.
_read_config_data
(
path
)
# if NOT using _read_config_data you should call set_options directly,
# to process any defined configuration for this plugin,
# if you don't define any options you can skip
#self.set_options()
# example consuming options from inventory source
mysession
=
apilib
.
session
(
user
=
self
.
get_option
(
'api_user'
),
password
=
self
.
get_option
(
'api_pass'
),
server
=
self
.
get_option
(
'api_server'
)
)
# make requests to get data to feed into inventory
mydata
=
mysession
.
getitall
()
#parse data and create inventory objects:
for
colo
in
mydata
```

:
for
server
in
mydata
[
colo
][
'servers'
]:
self
.
inventory
.
add_host
(
server
[
'name'
])
self
.
inventory
.
set_variable
(
server
[
'name'
],
'ansible_host'
,
server
[
'external_ip'
])
The specifics will vary depending on API and structure returned. Remember that if you get an inventory source error or any other issue, you should
raise
AnsibleParserError
to let Ansible know that the source was invalid or the process failed.
For examples on how to implement an inventory plugin, see the source code here:
lib/ansible/plugins/inventory
.
inventory object
?
The
inventory
object passed to
parse
has helpful methods for populating inventory.
add_group
adds a group to inventory if it doesn?t already exist. It takes the group name as the only positional argument.
add_child

adds a group or host that exists in inventory to a parent group in inventory. It takes two positional arguments, the name of the parent group and the name of the child group or host.
add_host
adds a host to inventory if it doesn?t already exist, optionally to a specific group. It takes the host name as the first argument and accepts two optional keyword arguments,
group
and
port
.
group
is the name of a group in inventory, and
port
is an integer.
set_variable
adds a variable to a group or host in inventory. It takes three positional arguments: the name of the group or host, the name of the variable, and the value of the variable.
To create groups and variables using Jinja2 expressions, see the section on implementing
constructed
features below.
To see other inventory object methods, see the source code here:
lib/ansible/inventory/data.py
.
inventory cache
?
To cache the inventory, extend the inventory plugin documentation with the inventory_cache documentation fragment and use the Cacheable base class.
extends_documentation_fragment
:
-
inventory_cache
class
InventoryModule
(
BaseInventoryPlugin
,
Constructable
,
Cacheable
):
NAME
=
'myplugin'
Next, load the cache plugin specified by the user to read from and update the cache. If your inventory plugin uses YAML-based configuration files and the
_read_config_data
method, the cache plugin is loaded within that method. If your inventory plugin does not use
_read_config_data
, you must load the cache explicitly with
load_cache_plugin
.
NAME
=
'myplugin'
def

```
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):
super
(
InventoryModule
,
self
)
.
parse
(
inventory
,
loader
,
path
)
self
.
load_cache_plugin
()
```

Before using the cache plugin, you must retrieve a unique cache key by using the
`get_cache_key`
method. This task needs to be done by all inventory modules using the cache, so that you don?t use/overwrite other parts of the cache.

```
def
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):
super
(
```

InventoryModule
,
self
)
.
parse
(
inventory
,
loader
,
path
)
self
.
load_cache_plugin
()
cache_key
=
self
.
get_cache_key
(
path
)
Now that you?ve enabled caching, loaded the correct plugin, and retrieved a unique cache key, you can set up the flow of data between the cache and your inventory using the
cache
parameter of the
parse
method. This value comes from the inventory manager and indicates whether the inventory is being refreshed (such as by the
--flush-cache
or the meta task
refresh_inventory
). Although the cache shouldn?t be used to populate the inventory when being refreshed, the cache should be updated with the new inventory if the user has enabled caching. You can use
self._cache
like a dictionary. The following pattern allows refreshing the inventory to work in conjunction with caching.
def
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):

```
super
(
InventoryModule
,
self
)
.
parse
(
inventory
,
loader
,
path
)
self
.
load_cache_plugin
()
cache_key
=
self
.
get_cache_key
(
path
)
# cache may be True or False at this point to indicate if the inventory is being refreshed
# get the user's cache option too to see if we should save the cache if it is changing
user_cache_setting
=
self
.
get_option
(
'cache'
)
# read if the user has caching enabled and the cache isn't being refreshed
attempt_to_read_cache
=
user_cache_setting
and
cache
# update if the user has caching enabled and the cache is being refreshed; update this value to True if the cache has expired below
cache_needs_update
=
user_cache_setting
and
not
cache
# attempt to read the cache if inventory isn't being refreshed and the user has caching enabled
if
attempt_to_read_cache
```

:
try
:
results
=
self
.
_cache
[
cache_key
]
except
KeyError
:
This occurs if the cache_key is not in the cache or if the cache_key expired, so the cache needs to be updated
cache_needs_update
=
True
if
not
attempt_to_read_cache
or
cache_needs_update
:
parse the provided inventory source
results
=
self
.
get_inventory
()
if
cache_needs_update
:
self
.
_cache
[
cache_key
]
=
results
submit the parsed data to the inventory object (add_host, set_variable, etc)
self
.
populate
(
results
)
After the
parse
method is complete, the contents of
self._cache
is used to set the cache plugin if the contents of the cache have changed.

You have three other cache methods available:
set_cache_plugin
forces the cache plugin to be set with the contents of
self._cache
, before the
parse
method completes
update_cache_if_changed
sets the cache plugin only if
self._cache
has been modified, before the
parse
method completes
clear_cache
flushes the cache, ultimately by calling the cache plugin?s
flush()
method, whose implementation is dependent upon the particular cache plugin in use. Note that if the user is using the same cache backend for facts and inventory, both will get flushed. To avoid this, the user can specify a distinct cache backend in their inventory plugin configuration.
constructed features
?
Inventory plugins can create host variables and groups from Jinja2 expressions and variables by using features from the
constructed
inventory plugin. To do this, use the
Constructable
base class and extend the inventory plugin?s documentation with the
constructed
documentation fragment.
extends_documentation_fragment
:
-
constructed
class
InventoryModule
(
BaseInventoryPlugin
,
Constructable
):
NAME
=
'ns.coll.myplugin'
There are three main options in the
constructed
documentation fragment:
compose
creates variables using Jinja2 expressions. This is implemented by calling the
_set_composite_vars
method.
keyed_groups
creates groups of hosts based on variable values. This is implemented by calling the
_add_host_to_keyed_groups
method.
groups

creates groups based on Jinja2 conditionals. This is implemented by calling the
_add_host_to_composed_groups
method.
Each method should be called for every host added to inventory. Three positional arguments are required: the constructed option, a dictionary of variables, and a host name. Calling the method
_set_composite_vars
first will allow
keyed_groups
and
groups
to use the composed variables.
By default, undefined variables are ignored. This is permitted by default for
compose
so you can make the variable definitions depend on variables that will be populated later in a play from other sources. For groups, it allows using variables that are not always present without having to use the
default
filter. To support configuring undefined variables to be an error, pass the constructed option
strict
to each of the methods as a keyword argument.
keyed_groups
and
groups
use any variables already associated with the host (for example, from an earlier inventory source).
_add_host_to_keyed_groups
and
add_host_to_composed_groups
can turn this off by passing the keyword argument
fetch_hostvars
.
Here is an example using all three methods:
```
def
add_host
(
self
,
hostname
,
host_vars
):
self
.
inventory
.
add_host
(
hostname
,
group
=
'all'
)
for
var_name
,
```

```
var_value
in
host_vars
.
items
():
self
.
inventory
.
set_variable
(
hostname
,
var_name
,
var_value
)
strict
=
self
.
get_option
(
'strict'
)
# Add variables created by the user's Jinja2 expressions to the host
self
.
_set_composite_vars
(
self
.
get_option
(
'compose'
),
host_vars
,
hostname
,
strict
=
True
)
# Create user-defined groups using variables and Jinja2 conditionals
self
.
_add_host_to_composed_groups
(
self
.
get_option
(
```

'groups'
),
host_vars
,
hostname
,
strict
=
strict
)
self
.
_add_host_to_keyed_groups
(
self
.
get_option
(
'keyed_groups'
),
host_vars
,
hostname
,
strict
=
strict
)
By default, group names created with
_add_host_to_composed_groups()
and
_add_host_to_keyed_groups()
are valid Python identifiers. Invalid characters are replaced with an underscore
_
. A plugin can change the sanitization used for the constructed features by setting
self._sanitize_group_name
to a new function. The core engine also does sanitization, so if the custom function is less strict it should be used in conjunction with the configuration setting
TRANSFORM_INVALID_GROUP_CHARS
.
from
ansible.inventory.group
import
to_safe_group_name
class
InventoryModule
(
BaseInventoryPlugin
,
Constructable
):
NAME
=
'ns.coll.myplugin'

```
@staticmethod
def
custom_sanitizer
(
name
):
return
to_safe_group_name
(
name
,
replacer
=
"
)
def
parse
(
self
,
inventory
,
loader
,
path
,
cache
=
True
):
super
(
InventoryModule
,
self
)
.
parse
(
inventory
,
loader
,
path
)
self
.
_sanitize_group_name
=
custom_sanitizer
```

Common format for inventory sources

?

To simplify development, most plugins use a standard YAML-based configuration file as the inventory source. The file has only one required field

plugin
, which should contain the name of the plugin that is expected to consume the file.
Depending on other common features used, you might need other fields, and you can add custom options in each plugin as required.
For example, if you use the integrated caching,
cache_plugin
,
cache_timeout
and other cache-related fields could be present.
The ?auto? plugin
?
From Ansible 2.5 onwards, we include the
auto inventory plugin
and enable it by default. If the
plugin
field in your standard configuration file matches the name of your inventory plugin, the
auto
inventory plugin will load your plugin. The ?auto? plugin makes it easier to use your plugin without having to update configurations.
Inventory scripts
?
Even though we now have inventory plugins, we still support inventory scripts, not only for backwards compatibility but also to allow users to use other programming languages.
Inventory script conventions
?
Inventory scripts must accept the
--list
and
--host
<hostname>
arguments. Although other arguments are allowed, Ansible will not use them.
Such arguments might still be useful for executing the scripts directly.
When the script is called with the single argument
--list
, the script must output to stdout a JSON object that contains all the groups to be managed. Each group?s value should be either an object containing a list of each host, any child groups, and potential group variables, or simply a list of hosts:
{
"group001"
:
{
"hosts"
:
[
"host001"
,
"host002"
],
"vars"
:
{
"var1"
:
true

},
"children"
:
[
"group002"
]
},
"group002"
:
{
"hosts"
:
[
"host003"
,
"host004"
],
"vars"
:
{
"var2"
:
500
},
"children"
:[]
}
}
If any of the elements of a group are empty, they may be omitted from the output.
When called with the argument
--host
<hostname>
(where <hostname> is a host from above), the script must print a JSON object, either empty or containing variables to make them available to templates and playbooks. For example:
{
"VAR001"
:
"VALUE"
,
"VAR002"
:
"VALUE"
}
Printing variables is optional. If the script does not print variables, it should print an empty JSON object.
Tuning the external inventory script
?
New in version 1.3.
The stock inventory script system mentioned above works for all versions of Ansible, but calling
--host
for every host can be rather inefficient, especially if it involves API calls to a remote subsystem.
To avoid this inefficiency, if the inventory script returns a top-level element called ?_meta?, it is possible to return all the host variables in a single script execution. When this meta element contains a value for ?hostvars?, the inventory script will not be invoked with
--host

for each host. This behavior results in a significant performance increase for large numbers of hosts.
The data to be added to the top-level JSON object looks like this:

```
{

    # results of inventory script as above go here
    # ...

    "_meta": {
        "hostvars": {
            "host001": {
                "var001" : "value"
            },
            "host002": {
                "var002": "value"
            }
        }
    }
}
```

To satisfy the requirements of using
_meta
, to prevent ansible from calling your inventory with
--host
you must at least populate
_meta
with an empty
hostvars
object.
For example:

```
{

    # results of inventory script as above go here
    # ...

    "_meta": {
        "hostvars": {}
    }
}
```

If you intend to replace an existing static inventory file with an inventory script, it must return a JSON object which contains an ?all? group that includes every host in the inventory as a member and every group in the inventory as a child. It should also include an ?ungrouped? group which contains all hosts which are not members of any other group.
A skeleton example of this JSON object is:

```
{
"_meta"
:
{
"hostvars"
:
{}
},
"all"
:
{
"children"
:
```

[
"ungrouped"
]
},
"ungrouped"
:
{
"children"
:
[
]
}
}
An easy way to see how this should look is using
ansible-inventory
, which also supports
--list
and
--host
parameters like an inventory script would.
See also
Python API
Python API to Playbooks and Ad Hoc Task Execution
Developing modules
Get started with developing a module
Developing plugins
How to develop plugins
AWX
REST API endpoint and GUI for Ansible, syncs with dynamic inventory
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_locally.html

Adding modules and plugins locally ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Edit on GitHub
Adding modules and plugins locally
?
You can extend Ansible by adding custom modules or plugins. You can create them from scratch or copy existing ones for local use. You can store a local module or plugin on your Ansible control node and share it with your team or organization. You can also share plugins and modules by including them in a collection, then publishing the collection on Ansible Galaxy.
If you are using a local module or plugin but Ansible cannot find it, this page is all you need.
If you want to create a plugin or a module, see
Developing plugins
,
Developing modules
and
Developing collections
.
Extending Ansible with local modules and plugins offers shortcuts such as:
You can copy other people?s modules and plugins.
When writing a new module, you can choose any programming language you like.
You do not have to clone any repositories.
You do not have to open a pull request.
You do not have to add tests (though we recommend that you do!).

Modules and plugins: what is the difference?
?
If you are looking to add functionality to Ansible, you might wonder whether you need a module or a plugin. Here is a

quick overview to help you understand what you need:
Plugins
extend Ansible?s core functionality. Most plugin types execute on the control node within the
/usr/bin/ansible
process. Plugins offer options and extensions for the core features of Ansible: transforming data, logging output, connecting to inventory, and more.
Modules are a type of plugin that execute automation tasks on a ?target? (usually a remote system). Modules work as standalone scripts that Ansible executes in their own process outside of the control node. Modules interface with Ansible mostly with JSON, accepting arguments and returning information by printing a JSON string to stdout before exiting. Unlike the other plugins (which must be written in Python), modules can be written in any language; although Ansible provides modules in Python and Powershell only.
Adding modules and plugins in collections
?
You can add modules and plugins by
creating a collection
. With a collection, you can use custom modules and plugins in any playbook or role. You can share your collection easily at any time through Ansible Galaxy.
The rest of this page describes other methods of using local, standalone modules or plugins.
Adding a module or plugin outside of a collection
?
You can configure Ansible to load standalone local modules or plugins in specific locations and make them available to all playbooks and roles (using configured paths). Alternatively, you can make a non-collection local module or plugin available only to certain playbooks or roles (with adjacent paths).
Adding standalone local modules for all playbooks and roles
?
To load standalone local modules automatically and make them available to all playbooks and roles, use the
DEFAULT_MODULE_PATH
configuration setting or the
ANSIBLE_LIBRARY
environment variable. The configuration setting and environment variable take a colon-separated list, similar to
$PATH
. You have two options:
Add your standalone local module to one of the default configured locations. See the
DEFAULT_MODULE_PATH
configuration setting for details. Default locations may change without notice.
Add the location of your standalone local module to an environment variable or configuration:
the
ANSIBLE_LIBRARY
environment variable
the
DEFAULT_MODULE_PATH
configuration setting
To view your current configuration settings for modules:
ansible-config dump |grep DEFAULT_MODULE_PATH
After you save your module file in one of these locations, Ansible loads it and you can use it in any local task, playbook, or role.
To confirm that
my_local_module
is available:
type
ansible
localhost
-m
my_local_module

to see the output for that module, or
type
ansible-doc
-t
module
my_local_module
to see the documentation for that module
Note
This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.
Note
The
ansible-doc
command can parse module documentation from modules written in Python or an adjacent YAML file. If you have a module written in a programming language other than Python, you should write the documentation in a Python or YAML file adjacent to the module file.
Adjacent YAML documentation files
Adding standalone local modules for selected playbooks or a single role
?
Ansible automatically loads all executable files from certain directories adjacent to your playbook or role as modules. Standalone modules in these locations are available only to the specific playbook, playbooks, or role in the parent directory.
To use a standalone module only in a selected playbook or playbooks, store the module in a subdirectory called
library
in the directory that contains the playbook or playbooks.
To use a standalone module only in a single role, store the module in a subdirectory called
library
within that role.
Note
This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.
Warning
Roles contained in collections cannot contain any modules or other plugins. All plugins in a collection must live in the collection
plugins
directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new modules, we recommend distributing them in
collections
, not in roles.
Adding a non-module plugin locally outside of a collection
?
You can configure Ansible to load standalone local plugins in a specified location or locations and make them available to all playbooks and roles. Alternatively, you can make a standalone local plugin available only to specific playbooks or roles.
Note
Although modules are plugins, the naming patterns for directory names and environment variables that apply to other plugin types do not apply to modules. See
Adding a module or plugin outside of a collection
.
Adding local non-module plugins for all playbooks and roles
?
To load standalone local plugins automatically and make them available to all playbooks and roles, use the configuration setting or environment variable for the type of plugin you are adding. These configuration settings and environment variables take a colon-separated list, similar to

$PATH
. You have two options:
Add your local plugin to one of the default configured locations. See
configuration settings
for details on the correct configuration setting for the plugin type. Default locations may change without notice.
Add the location of your local plugin to an environment variable or configuration:
the relevant
ANSIBLE_plugin_type_PLUGINS
environment variable - for example,
$ANSIBLE_INVENTORY_PLUGINS
or
$ANSIBLE_VARS_PLUGINS
the relevant
plugin_type_PATH
configuration setting, most of which begin with
DEFAULT_
- for example,
DEFAULT_CALLBACK_PLUGIN_PATH
or
DEFAULT_FILTER_PLUGIN_PATH
or
BECOME_PLUGIN_PATH
To view your current configuration settings for non-module plugins:
ansible-config dump |grep plugin_type_PATH
After your plugin file is added to one of these locations, Ansible loads it and you can use it in any local module, task, playbook, or role. For more information on environment variables and configuration settings, see
Ansible Configuration Settings
.
To confirm that
plugins/plugin_type/my_local_plugin
is available:
type
ansible-doc
-t
<plugin_type>
my_local_lookup_plugin
to see the documentation for that plugin - for example,
ansible-doc
-t
lookup
my_local_lookup_plugin
The
ansible-doc
command works for most plugin types, but not for action, filter, or test plugins. See
ansible-doc
for more details.
Adding standalone local plugins for selected playbooks or a single role
?
Ansible automatically loads all plugins from certain directories adjacent to your playbook or role, loading each type of plugin separately from a directory named for the type of plugin. Standalone plugins in these locations are available only to the specific playbook, playbooks, or role in the parent directory.
To use a standalone plugin only in a selected playbook or playbooks, store the plugin in a subdirectory for the correct
plugin_type
(for example,

callback_plugins
or
inventory_plugins
) in the directory that contains the playbooks. These directories must use the
_plugins
suffix. For a full list of plugin types, see
Working with plugins
.
To use a standalone plugin only in a single role, store the plugin in a subdirectory for the correct
plugin_type
(for example,
cache_plugins
or
strategy_plugins
) within that role. When shipped as part of a role, the plugin is available as soon as the role is executed. These directories must use the
_plugins
suffix. For a full list of plugin types, see
Working with plugins
.
Warning
Roles contained in collections cannot contain any plugins. All plugins in a collection must live in the collection
plugins
directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new plugins, we recommend distributing them in
collections
, not in roles.
Warning
Some plugin types are needed early during Ansible execution, such as callbacks, inventory, and cache. These plugin types cannot be loaded dynamically and must exist in configured paths or be referenced by FQCN in configuration.
Using
ansible.legacy
to access custom versions of an
ansible.builtin
module
?
If you need to override one of the
ansible.builtin
modules and are using FQCN, you need to use
ansible.legacy
as part of the fully-qualified collection name (FQCN). For example, if you had your own
copy
module, you would access it as
ansible.legacy.copy
. See
Using ansible.legacy to access local custom modules from collections-based roles
for details on how to use custom modules with roles within a collection.

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_modules_best_practices.html

Conventions, tips, and pitfalls ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

As you design and develop modules, follow these basic conventions and tips for clean, usable code:

Scoping your module(s)

?

Especially if you want to contribute your module(s) to an existing Ansible Collection, make sure each module includes enough logic and functionality, but not too much. If these guidelines seem confusing, consider
whether you really need to write a module
at all.

Each module should have a concise and well-defined functionality. Basically, follow the UNIX philosophy of doing one thing well.

Do not add
get
,
list
or
info
state options to an existing module - create a new
_info
or
_facts
module.

Modules should not require that a user know all the underlying options of an API/tool to be used. For example, if the legal values for a required module option cannot be documented, the module does not belong in Ansible Core.

Modules should encompass much of the logic for interacting with a resource. A lightweight wrapper around a complex API forces users to offload too much logic into their playbooks. If you want to connect Ansible to a complex API,
create multiple modules
that interact with smaller individual pieces of the API.
Avoid creating a module that does the work of other modules; this leads to code duplication and divergence, and makes things less uniform, unpredictable and harder to maintain. Modules should be the building blocks. If you are asking ?how can I have a module execute other modules? ? you want to write a role.
Designing module interfaces
?
If your module is addressing an object, the option for that object should be called
name
whenever possible, or accept
name
as an alias.
Modules accepting boolean status should accept
yes
,
no
,
true
,
false
, or anything else a user may likely throw at them. The AnsibleModule common code supports this with
type='bool'
.
Avoid
action
/
command
, they are imperative and not declarative, there are other ways to express the same thing.
General guidelines & tips
?
Each module should be self-contained in one file, so it can be auto-transferred by
ansible-core
.
Module name MUST use underscores instead of hyphens or spaces as a word separator. Using hyphens and spaces will prevent
ansible-core
from importing your module.
Always use the
hacking/test-module.py
script when developing modules - it will warn you about common pitfalls.
If you have a local module that returns information specific to your installations, a good name for this module is
site_info
.
Eliminate or minimize dependencies. If your module has dependencies, document them at the top of the module file and raise JSON error messages when dependency import fails.
Don?t write to files directly; use a temporary file and then use the
atomic_move
function from
ansible.module_utils.basic
to move the updated temporary file into place. This prevents data corruption and ensures that the correct context for the file is kept.
Avoid creating caches. Ansible is designed without a central server or authority, so you cannot guarantee it will not run

with different permissions, options or locations. If you need a central authority, have it on top of Ansible (for example, using bastion/cm/ci server, AWX, or the Red Hat Ansible Automation Platform); do not try to build it into modules.
If you package your module(s) in an RPM, install the modules on the control machine in
/usr/share/ansible
. Packaging modules in RPMs is optional.
Functions and Methods
?
Each function should be concise and should describe a meaningful amount of work.
?Don?t repeat yourself? is generally a good philosophy.
Function names should use underscores:
my_function_name
.
The name of each function should describe what the function does.
Each function should have a docstring.
If your code is too nested, that?s usually a sign the loop body could benefit from being a function. Parts of our existing code are not the best examples of this at times.
Python tips
?
Include a
main
function that wraps the normal execution.
Call your
main
function from a conditional so you can import it into unit tests - for example:
if
__name__
==
'__main__'
:
main
()
Importing and using shared code
?
Use shared code whenever possible - don?t reinvent the wheel. Ansible offers the
AnsibleModule
common Python code, plus
utilities
for many common use cases and patterns. You can also create documentation fragments for docs that apply to multiple modules.
Import
ansible.module_utils
code in the same place as you import other libraries.
Do NOT use wildcards (*) for importing other python modules; instead, list the function(s) you are importing (for example,
from
some.other_python_module.basic
import
otherFunction
).
Import custom packages in
try
/
except
, capture any import errors, and handle them with

fail_json()
in
main()
. For example:
import
traceback
from
ansible.module_utils.basic
import
missing_required_lib
LIB_IMP_ERR
=
None
try
:
import
foo
HAS_LIB
=
True
except
:
HAS_LIB
=
False
LIB_IMP_ERR
=
traceback
.
format_exc
()
Then in
main()
, just after the argspec, do
if
not
HAS_LIB
:
module
.
fail_json
(
msg
=
missing_required_lib
(
"foo"
),
exception
=
LIB_IMP_ERR
)
And document the dependency in the
requirements

section of your module?s
DOCUMENTATION block
.
Handling module failures
?
When your module fails, help users understand what went wrong. If you are using the
AnsibleModule
common Python code, the
failed
element will be included for you automatically when you call
fail_json
. For polite module failure behavior:
Include a key of
failed
along with a string explanation in
msg
. If you don?t do this, Ansible will use standard return codes: 0=success and non-zero=failure.
Don?t raise a traceback (stacktrace). Ansible can deal with stacktraces and automatically converts anything unparsable into a failed result, but raising a stacktrace on module failure is not user-friendly.
Do not use
sys.exit()
. Use
fail_json()
from the module object.
Handling exceptions (bugs) gracefully
?
Validate upfront?fail fast and return useful and clear error messages.
Use defensive programming?use a simple design for your module, handle errors gracefully, and avoid direct stacktraces.
Fail predictably?if we must fail, do it in a way that is the most expected. Either mimic the underlying tool or the general way the system works.
Give out a useful message on what you were doing and add exception messages to that.
Avoid catchall exceptions, they are not very useful unless the underlying API gives very good error messages pertaining the attempted action.
Creating correct and informative module output
?
Modules must output valid JSON only. Follow these guidelines for creating correct, useful module output:
Module return data must be encoded as strict UTF-8. Modules that cannot return UTF-8 encoded data should return the data encoded by something such as base64. Optionally modules can make the determination if they can encode as UTF-8 and utilize
errors='replace'
to replace non UTF-8 characters making the return values lossy.
Make your top-level return type a hash (dictionary).
Nest complex return values within the top-level hash.
Incorporate any lists or simple scalar values within the top-level return hash.
Do not send module output to standard error, because the system will merge standard out with standard error and prevent the JSON from parsing.
Capture standard error and return it as a variable in the JSON on standard out. This is how the command module is implemented.
Never do
print("some
status
message")
in a module, because it will not produce valid JSON output.

Always return useful data, even when there is no change.
Be consistent about returns (some modules are too random), unless it is detrimental to the state/action.
Make returns reusable?most of the time you don?t want to read it, but you do want to process it and re-purpose it.
Return diff if in diff mode. This is not required for all modules, as it won?t make sense for certain ones, but please include it when applicable.
Enable your return values to be serialized as JSON with Python?s standard
JSON encoder and decoder
library. Basic python types (strings, int, dicts, lists, and so on) are serializable.
Do not return an object using exit_json(). Instead, convert the fields you need from the object into the fields of a dictionary and return the dictionary.
Results from many hosts will be aggregated at once, so your module should return only relevant output. Returning the entire contents of a log file is generally bad form.
If a module returns stderr or otherwise fails to produce valid JSON, the actual output will still be shown in Ansible, but the command will not succeed.
Following Ansible conventions
?
Ansible conventions offer a predictable user interface across all modules, playbooks, and roles. To follow Ansible conventions in your module development:
Use consistent names across modules (yes, we have many legacy deviations - don?t make the problem worse!).
Use consistent options (arguments) within your module(s).
Do not use ?message? or ?syslog_facility? as an option name, because this is used internally by Ansible.
Normalize options with other modules - if Ansible and the API your module connects to use different names for the same option, add aliases to your options so the user can choose which names to use in tasks and playbooks.
Return facts from
*_facts
modules in the
ansible_facts
field of the
result dictionary
so other modules can access them.
Implement
check_mode
in all
*_info
and
*_facts
modules. Playbooks which conditionalize based on fact information will only conditionalize correctly in
check_mode
if the facts are returned in
check_mode
. Usually you can add
supports_check_mode=True
when instantiating
AnsibleModule
.
Use module-specific environment variables. For example, if you use the helpers in
module_utils.api
for basic authentication with
module_utils.urls.fetch_url()
and you fall back on environment variables for default values, use a module-specific environment variable like
API_<MODULENAME>_USERNAME
to avoid conflicts between modules.
Keep module options simple and focused - if you?re loading a lot of choices/states on an existing option, consider adding a new, simple option instead.

Keep options small when possible. Passing a large data structure to an option might save us a few tasks, but it adds a complex requirement that we cannot easily validate before passing on to the module.
If you want to pass complex data to an option, write an expert module that allows this, along with several smaller modules that provide a more ?atomic? operation against the underlying APIs and services. Complex operations require complex data. Let the user choose whether to reflect that complexity in tasks and plays or in vars files.
Implement declarative operations (not CRUD) so the user can ignore existing state and focus on final state. For example, use
started/stopped
,
present/absent
.
Strive for a consistent final state (aka idempotency). If running your module twice in a row against the same system would result in two different states, see if you can redesign or rewrite to achieve consistent final state. If you can?t, document the behavior and the reasons for it.
Provide consistent return values within the standard Ansible return structure, even if NA/None are used for keys normally returned under other options.
Module Security
?
Avoid passing user input from the shell.
Always check return codes.
You must always use
module.run_command
, not
subprocess
or
Popen
or
os.system
.
Avoid using the shell unless absolutely necessary.
If you must use the shell, you must pass
use_unsafe_shell=True
to
module.run_command
.
If any variables in your module can come from user input with
use_unsafe_shell=True
, you must wrap them with
pipes.quote(x)
.
When fetching URLs, use
fetch_url
or
open_url
from
ansible.module_utils.urls
. Do not use
urllib2
, which does not natively verify TLS certificates and so is insecure for https.
Sensitive values marked with
no_log=True
will automatically have that value stripped from module return values. If your module could return these sensitive values as part of a dictionary key name, you should call the
ansible.module_utils.basic.sanitize_keys()

function to strip the values from the keys. See the
uri
module for an example.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_modules_general_windows.html

Windows module development walkthrough ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Windows module development walkthrough
Edit on GitHub
Windows module development walkthrough
?
In this section, we will walk through developing, testing, and debugging an Ansible Windows module.
Because Windows modules are written in Powershell and need to be run on a Windows host, this guide differs from the usual development walkthrough guide.
What?s covered in this section:
Windows environment setup
Create a Windows server in a VM
Create an Ansible inventory
Provisioning the environment
Windows new module development
Windows module utilities
Exposing shared module options
Windows playbook module testing
Windows debugging
Windows unit testing
Windows integration testing
Windows communication and development support
Windows environment setup
?
Unlike Python module development which can be run on the host that runs Ansible, Windows modules need to be written and tested for Windows hosts.
While evaluation editions of Windows can be downloaded from Microsoft, these images are usually not ready to be used by Ansible without further modification. The easiest way to set up a Windows host so that it is ready to by used by Ansible is to set up a virtual machine using Vagrant.
Vagrant can be used to download existing OS images called boxes that are then deployed to a hypervisor like VirtualBox. These boxes can either be created and stored offline or they can be downloaded from a central repository called Vagrant Cloud.
This guide will use the Vagrant boxes created by the packer-windoze repository which have also been uploaded to

Vagrant Cloud
.
To find out more info on how these images are created, please go to the GitHub
repo and look at the
README
file.
Before you can get started, the following programs must be installed (please consult the Vagrant and
VirtualBox documentation for installation instructions):
Vagrant
VirtualBox
Create a Windows server in a VM
?
To create a single Windows Server 2016 instance, run the following:
vagrant
init
jborean93/WindowsServer2016
vagrant
up
This will download the Vagrant box from Vagrant Cloud and add it to the local
boxes on your host and then start up that instance in VirtualBox. When starting
for the first time, the Windows VM will run through the sysprep process and
then create a HTTP and HTTPS WinRM listener automatically. Vagrant will finish
its process once the listeners are online, after which the VM can be used by Ansible.
Create an Ansible inventory
?
The following Ansible inventory file can be used to connect to the newly
created Windows VM:
[windows]
WindowsServer ansible_host
=
127.0.0.1
[windows:vars]
ansible_user
=
vagrant
ansible_password
=
vagrant
ansible_port
=
55986
ansible_connection
=
winrm
ansible_winrm_transport
=
ntlm
ansible_winrm_server_cert_validation
=
ignore
Note
The port
55986
is automatically forwarded by Vagrant to the

Windows host that was created, if this conflicts with an existing local
port then Vagrant will automatically use another one at random and display
show that in the output.
The OS that is created is based on the image set. The following
images can be used:
jborean93/WindowsServer2012
jborean93/WindowsServer2012R2
jborean93/WindowsServer2016
jborean93/WindowsServer2019
jborean93/WindowsServer2022
When the host is online, it can accessible by RDP on
127.0.0.1:3389
but the
port may differ depending if there was a conflict. To get rid of the host, run
vagrant
destroy
--force
and Vagrant will automatically remove the VM and
any other files associated with that VM.
While this is useful when testing modules on a single Windows instance, these
host won?t work without modification with domain based modules. The Vagrantfile
at
ansible-windows
can be used to create a test domain environment to be used in Ansible. This
repo contains three files which are used by both Ansible and Vagrant to create
multiple Windows hosts in a domain environment. These files are:
Vagrantfile
: The Vagrant file that reads the inventory setup of
inventory.yml
and provisions the hosts that are required
inventory.yml
: Contains the hosts that are required and other connection information such as IP addresses and forwarded ports
main.yml
: Ansible playbook called by Vagrant to provision the domain control node and join the child hosts to the domain
By default, these files will create the following environment:
A single AD domain controller running on Windows Server 2016
Five child hosts for each major Windows Server version joined to that domain
A domain with the DNS name
domain.local
A local administrator account on each host with the username
vagrant
and password
vagrant
A domain admin account
vagrant-domain@domain.local
with the password
VagrantPass1
The domain name and accounts can be modified by changing the variables
domain_*
in the
inventory.yml
file if it is required. The inventory
file can also be modified to provision more or less servers by changing the
hosts that are defined under the

domain_children
key. The host variable
ansible_host
is the private IP that will be assigned to the VirtualBox host
only network adapter while
vagrant_box
is the box that will be used to
create the VM.
Provisioning the environment
?
To provision the environment as is, run the following:
git
clone
https://github.com/jborean93/ansible-windows.git
cd
vagrant
vagrant
up
Note
Vagrant provisions each host sequentially so this can take some time
to complete. If any errors occur during the Ansible phase of setting up the
domain, run
vagrant
provision
to rerun just that step.
Unlike setting up a single Windows instance with Vagrant, these hosts can also
be accessed using the IP address directly as well as through the forwarded
ports. It is easier to access it over the host only network adapter as the
normal protocol ports are used, for example RDP is still over
3389
. In cases where
the host cannot be resolved using the host only network IP, the following
protocols can be access over
127.0.0.1
using these forwarded ports:
RDP
: 295xx
SSH
: 296xx
WinRM
HTTP
: 297xx
WinRM
HTTPS
: 298xx
SMB
: 299xx
Replace
xx
with the entry number in the inventory file where the domain
controller started with
00
and is incremented from there. For example, in
the default

inventory.yml
file, WinRM over HTTPS for
SERVER2012R2
is
forwarded over port
29804
as it is the fourth entry in
domain_children
.
Windows new module development
?
When creating a new module there are a few things to keep in mind:
Module code is in Powershell (.ps1) files while the documentation is contained in Python (.py) files of the same name
Avoid using
Write-Host/Debug/Verbose/Error
in the module and add what needs to be returned to the
$module.Result
variable
To fail a module, call
$module.FailJson("failure
message
here")
, an Exception or ErrorRecord can be set to the second argument for a more descriptive error message
You can pass in the exception or ErrorRecord as a second argument to
FailJson("failure",
$_)
to get a more detailed output
Most new modules require check mode and integration tests before they are merged into the main Ansible codebase
Avoid using try/catch statements over a large code block, rather use them for individual calls so the error message can be more descriptive
Try and catch specific exceptions when using try/catch statements
Avoid using PSCustomObjects unless necessary
Look for common functions in
./lib/ansible/module_utils/powershell/
and use the code there instead of duplicating work. These can be imported by adding the line
#Requires
-Module
*
where * is the file name to import, and will be automatically included with the module code sent to the Windows target when run through Ansible
As well as PowerShell module utils, C# module utils are stored in
./lib/ansible/module_utils/csharp/
and are automatically imported in a module execution if the line
#AnsibleRequires
-CSharpUtil
*
is present
C# and PowerShell module utils achieve the same goal but C# allows a developer to implement low level tasks, such as calling the Win32 API, and can be faster in some cases
Ensure the code runs under Powershell v5.1 and higher on Windows Server 2016 and higher; if higher minimum Powershell or OS versions are required, ensure the documentation reflects this clearly
Ansible runs modules under strictmode version 2.0. Be sure to test with that enabled by putting
Set-StrictMode
-Version

2.0
at the top of your dev script
Favor native Powershell cmdlets over executable calls if possible
Use the full cmdlet name instead of aliases, for example
Remove-Item
over
rm
Use named parameters with cmdlets, for example
Remove-Item
-Path
C:\temp
over
Remove-Item
C:\temp
A very basic Powershell module
win_environment
incorporates best practices for Powershell modules. It demonstrates how to implement check-mode and diff-support, and also shows a warning to the user when a specific condition is met.
A slightly more advanced module is
win_uri
which additionally shows how to use different parameter types (bool, str, int, list, dict, path) and a selection of choices for parameters, how to fail a module and how to handle exceptions.
As part of the new
AnsibleModule
wrapper, the input parameters are defined and validated based on an argument
spec. The following options can be set at the root level of the argument spec:
mutually_exclusive
: A list of lists, where the inner list contains module options that cannot be set together
no_log
: Stops the module from emitting any logs to the Windows Event log
options
: A dictionary where the key is the module option and the value is the spec for that option
required_by
: A dictionary where the option(s) specified by the value must be set if the option specified by the key is also set
required_if
: A list of lists where the inner list contains 3 or 4 elements;
The first element is the module option to check the value against
The second element is the value of the option specified by the first element, if matched then the required if check is run
The third element is a list of required module options when the above is matched
An optional fourth element is a boolean that states whether all module options in the third elements are required (default:
$false
) or only one (
$true
)
required_one_of
: A list of lists, where the inner list contains module options where at least one must be set
required_together
: A list of lists, where the inner list contains module options that must be set together
supports_check_mode
: Whether the module supports check mode, by default this is
$false
The actual input options for a module are set within the
options
value as a dictionary. The keys of this dictionary

are the module option names while the values are the spec of that module option. Each spec can have the following options set:
aliases
: A list of aliases for the module option
choices
: A list of valid values for the module option, if
type=list
then each list value is validated against the choices and not the list itself
default
: The default value for the module option if not set
deprecated_aliases
: A list of hashtables that define aliases that are deprecated and the versions they will be removed in. Each entry must contain the keys
name
and
collection_name
with either
version
or
date
elements
: When
type=list
, this sets the type of each list value, the values are the same as
type
no_log
: Will sanitize the input value before being returned in the
module_invocation
return value
removed_in_version
: States when a deprecated module option is to be removed, a warning is displayed to the end user if set
removed_at_date
: States the date (YYYY-MM-DD) when a deprecated module option will be removed, a warning is displayed to the end user if set
removed_from_collection
: States from which collection the deprecated module option will be removed; must be specified if one of
removed_in_version
and
removed_at_date
is specified
required
: Will fail when the module option is not set
type
: The type of the module option, if not set then it defaults to
str
. The valid types are;
bool
: A boolean value
dict
: A dictionary value, if the input is a JSON or key=value string then it is converted to dictionary
float
: A float or
Single
value

int
: An Int32 value
json
: A string where the value is converted to a JSON string if the input is a dictionary
list
: A list of values,
elements=<type>
can convert the individual list value types if set. If
elements=dict
then
options
is defined, the values will be validated against the argument spec. When the input is a string then the string is split by
,
and any whitespace is trimmed
path
: A string where values likes
%TEMP%
are expanded based on environment values. If the input value starts with
\\?\
then no expansion is run
raw
: No conversions occur on the value passed in by Ansible
sid
: Will convert Windows security identifier values or Windows account names to a
SecurityIdentifier
value
str
: The value is converted to a string
When
type=dict
, or
type=list
and
elements=dict
, the following keys can also be set for that module option:
apply_defaults
: The value is based on the
options
spec defaults for that key if
True
and null if
False
. Only valid when the module option is not defined by the user and
type=dict
.
mutually_exclusive
: Same as the root level
mutually_exclusive
but validated against the values in the sub dict
options
: Same as the root level
options
but contains the valid options for the sub option
required_if

: Same as the root level
required_if
but validated against the values in the sub dict
required_by
: Same as the root level
required_by
but validated against the values in the sub dict
required_together
: Same as the root level
required_together
but validated against the values in the sub dict
required_one_of
: Same as the root level
required_one_of
but validated against the values in the sub dict
A module type can also be a delegate function that converts the value to whatever is required by the module option. For example the following snippet shows how to create a custom type that creates a
UInt64
value:
$spec
=
@{
uint64_type
=
@{
type
=
[Func[[Object], [UInt64]]]
{
[System.UInt64]
::
Parse
(
$args
[
0
])
}
}
}
$uint64_type
=
$module
.
Params
.
uint64_type
When in doubt, look at some of the other core modules and see how things have been implemented there.
Sometimes there are multiple ways that Windows offers to complete a task; this is the order to favor when writing modules:
Native Powershell cmdlets like
Remove-Item
-Path

C:\temp
-Recurse
.NET classes like
[System.IO.Path]::GetRandomFileName()
WMI objects through the
New-CimInstance
cmdlet
COM objects through
New-Object
-ComObject
cmdlet
Calls to native executables like
Secedit.exe
PowerShell modules support a small subset of the
#Requires
options built
into PowerShell as well as some Ansible-specific requirements specified by
#AnsibleRequires
. These statements can be placed at any point in the script,
but are most commonly near the top. They are used to make it easier to state the
requirements of the module without writing any of the checks. Each
requires
statement must be on its own line, but there can be multiple requires statements
in one script.
These are the checks that can be used within Ansible modules:
#Requires
-Module
Ansible.ModuleUtils.<module_util>
: Added in Ansible 2.4, specifies a module_util to load in for the module execution.
#Requires
-Version
x.y
: Added in Ansible 2.5, specifies the version of PowerShell that is required by the module. The module will fail if this requirement is not met.
#AnsibleRequires
-PowerShell
<module_util>
: Added in Ansible 2.8, like
#Requires
-Module
, this specifies a module_util to load in for module execution.
#AnsibleRequires
-CSharpUtil
<module_util>
: Added in Ansible 2.8, specifies a C# module_util to load in for the module execution.
#AnsibleRequires
-OSVersion
x.y
: Added in Ansible 2.5, specifies the OS build version that is required by the module and will fail if this requirement is not met. The actual OS version is derived from
[Environment]::OSVersion.Version
.
#AnsibleRequires
-Become

: Added in Ansible 2.5, forces the exec runner to run the module with
become
, which is primarily used to bypass WinRM restrictions. If
ansible_become_user
is not specified then the
SYSTEM
account is used instead.
The
#AnsibleRequires
-PowerShell
and
#AnsibleRequires
-CSharpUtil
support further features such as:
Importing a util contained in a collection (added in Ansible 2.9)
Importing a util by relative names (added in Ansible 2.10)
Specifying the util is optional by adding
-Optional
to the import
declaration (added in Ansible 2.12).
See the below examples for more details:
Imports the PowerShell Ansible.ModuleUtils.Legacy provided by Ansible itself
#AnsibleRequires -PowerShell Ansible.ModuleUtils.Legacy
Imports the PowerShell my_util in the my_namesapce.my_name collection
#AnsibleRequires -PowerShell ansible_collections.my_namespace.my_name.plugins.module_utils.my_util
Imports the PowerShell my_util that exists in the same collection as the current module
#AnsibleRequires -PowerShell ..module_utils.my_util
Imports the PowerShell Ansible.ModuleUtils.Optional provided by Ansible if it exists.
If it does not exist then it will do nothing.
#AnsibleRequires -PowerShell Ansible.ModuleUtils.Optional -Optional
Imports the C# Ansible.Process provided by Ansible itself
#AnsibleRequires -CSharpUtil Ansible.Process
Imports the C# my_util in the my_namespace.my_name collection
#AnsibleRequires -CSharpUtil ansible_collections.my_namespace.my_name.plugins.module_utils.my_util
Imports the C# my_util that exists in the same collection as the current module
#AnsibleRequires -CSharpUtil ..module_utils.my_util
Imports the C# Ansible.Optional provided by Ansible if it exists.
If it does not exist then it will do nothing.
#AnsibleRequires -CSharpUtil Ansible.Optional -Optional
For optional require statements, it is up to the module code to then verify
whether the util has been imported before trying to use it. This can be done by
checking if a function or type provided by the util exists or not.
While both
#Requires
-Module
and
#AnsibleRequires
-PowerShell
can be
used to load a PowerShell module it is recommended to use
#AnsibleRequires
.
This is because
#AnsibleRequires

supports collection module utils, imports
by relative util names, and optional util imports.
C# module utils can reference other C# utils by adding the line
using
Ansible.<module_util>;
to the top of the script with all the other
using statements.
Windows module utilities
?
Like Python modules, PowerShell modules also provide a number of module
utilities that provide helper functions within PowerShell. These module_utils
can be imported by adding the following line to a PowerShell module:
#Requires -Module Ansible.ModuleUtils.Legacy
This will import the module_util at
./lib/ansible/module_utils/powershell/Ansible.ModuleUtils.Legacy.psm1
and enable calling all of its functions. As of Ansible 2.8, Windows module
utils can also be written in C# and stored at
lib/ansible/module_utils/csharp
.
These module_utils can be imported by adding the following line to a PowerShell
module:
#AnsibleRequires -CSharpUtil Ansible.Basic
This will import the module_util at
./lib/ansible/module_utils/csharp/Ansible.Basic.cs
and automatically load the types in the executing process. C# module utils can
reference each other and be loaded together by adding the following line to the
using statements at the top of the util:
using
Ansible.Become
;
There are special comments that can be set in a C# file for controlling the
compilation parameters. The following comments can be added to the script;
//AssemblyReference
-Name
<assembly
dll>
[-CLR
[Core|Framework]]
: The assembly DLL to reference during compilation, the optional
-CLR
flag can also be used to state whether to reference when running under .NET Core, Framework, or both (if omitted)
//NoWarn
-Name
<error
id>
[-CLR
[Core|Framework]]
: A compiler warning ID to ignore when compiling the code, the optional
-CLR
works the same as above. A list of warnings can be found at
Compiler errors
As well as this, the following pre-processor symbols are defined;
CORECLR
: This symbol is present when PowerShell is running through .NET Core

WINDOWS
: This symbol is present when PowerShell is running on Windows
UNIX
: This symbol is present when PowerShell is running on Unix
A combination of these flags help to make a module util interoperable on both
.NET Framework and .NET Core, here is an example of them in action:
#if CORECLR
using
Newtonsoft.Json
;
#else
using
System.Web.Script.Serialization
;
#endif
//AssemblyReference -Name Newtonsoft.Json.dll -CLR Core
//AssemblyReference -Name System.Web.Extensions.dll -CLR Framework
// Ignore error CS1702 for all .NET types
//NoWarn -Name CS1702
// Ignore error CS1956 only for .NET Framework
//NoWarn -Name CS1956 -CLR Framework
The following is a list of module_utils that are packaged with Ansible and a general description of what
they do:
ArgvParser: Utility used to convert a list of arguments to an escaped string compliant with the Windows argument parsing rules.
CamelConversion: Utility used to convert camelCase strings/lists/dicts to snake_case.
CommandUtil: Utility used to execute a Windows process and return the stdout/stderr and rc as separate objects.
FileUtil: Utility that expands on the
Get-ChildItem
and
Test-Path
to work with special files like
C:\pagefile.sys
.
Legacy: General definitions and helper utilities for Ansible module.
LinkUtil: Utility to create, remove, and get information about symbolic links, junction points and hard inks.
SID: Utilities used to convert a user or group to a Windows SID and vice versa.
For more details on any specific module utility and their requirements, please see the
Ansible
module utilities source code
.
PowerShell module utilities can be stored outside of the standard Ansible
distribution for use with custom modules. Custom module_utils are placed in a
folder called
module_utils
located in the root folder of the playbook or role
directory.
C# module utilities can also be stored outside of the standard Ansible distribution for use with custom modules. Like
PowerShell utils, these are stored in a folder called
module_utils
and the file name must end in the extension
.cs
, start with
Ansible.

and be named after the namespace defined in the util.
The below example is a role structure that contains two PowerShell custom module_utils called
Ansible.ModuleUtils.ModuleUtil1
,
Ansible.ModuleUtils.ModuleUtil2
, and a C# util containing the namespace
Ansible.CustomUtil
:
meta/
main.yml
defaults/
main.yml
module_utils/
Ansible.ModuleUtils.ModuleUtil1.psm1
Ansible.ModuleUtils.ModuleUtil2.psm1
Ansible.CustomUtil.cs
tasks/
main.yml
Each PowerShell module_util must contain at least one function that has been exported with
Export-ModuleMember
at the end of the file. For example
Export-ModuleMember
-Function
Invoke-CustomUtil
,
Get-CustomInfo
Exposing shared module options
?
PowerShell module utils can easily expose common module options that a module can use when building its argument spec.
This allows common features to be stored and maintained in one location and have those features used by multiple modules with minimal effort. Any new features or bugfixes added to one of these utils are then automatically used by the various modules that call that util.
An example of this would be to have a module util that handles authentication and communication against an API This util can be used by multiple modules to expose a common set of module options like the API endpoint, username, password, timeout, cert validation, and so on without having to add those options to each module spec.
The standard convention for a module util that has a shared argument spec would have
A
Get-<namespace.name.util
name>Spec
function that outputs the common spec for a module
It is highly recommended to make this function name be unique to the module to avoid any conflicts with other utils that can be loaded
The format of the output spec is a Hashtable in the same format as the
$spec
used for normal modules
A function that takes in an
AnsibleModule
object called under the
-Module
parameter which it can use to get the shared options
Because these options can be shared across various module it is highly recommended to keep the module option names and
aliases in the shared spec as specific as they can be. For example do not have a util option called

password
,
rather you should prefix it with a unique name like
acme_password
.
Warning
Failure to have a unique option name or alias can prevent the util being used by module that also use those names or aliases for its own options.
The following is an example module util called
ServiceAuth.psm1
in a collection that implements a common way for
modules to authentication with a service.

```
Invoke-MyServiceResource
{
[
CmdletBinding
()]
param
(
[
Parameter
(
Mandatory
=
$true
)]
[
ValidateScript
({
$_
.
GetType
().
FullName
-eq
'Ansible.Basic.AnsibleModule'
})]
$Module
,
[
Parameter
(
Mandatory
=
$true
)]
[String]
$ResourceId
,
[String]
$State
=
'present'
)
```

```
# Process the common module options known to the util
$params
=
@{
ServerUri
=
$Module
.
Params
.
my_service_url
}
if
(
$Module
.
Params
.
my_service_username
)
{
$params
.
Credential
=
Get-MyServiceCredential
}
if
(
$State
-eq
'absent'
)
{
Remove-MyService
@params
-ResourceId
$ResourceId
}
else
{
New-MyService
@params
-ResourceId
$ResourceId
}
}
Get-MyNamespaceMyCollectionServiceAuthSpec
{
# Output the util spec
@{
options
=
@{
```

```
my_service_url
=
@{
type
=
'str'
;
required
=
$true
}
my_service_username
=
@{
type
=
'str'
}
my_service_password
=
@{
type
=
'str'
;
no_log
=
$true
}
}
required_together
=
@(
,@(
'my_service_username'
,
'my_service_password'
)
)
}
}
$exportMembers
=
@{
Function
=
'Get-MyNamespaceMyCollectionServiceAuthSpec'
,
'Invoke-MyServiceResource'
}
Export-ModuleMember
@exportMembers
```

For a module to take advantage of this common argument spec it can be set out like

```
#!powershell
```

```
# Include the module util ServiceAuth.psm1 from the my_namespace.my_collection collection
#AnsibleRequires -PowerShell ansible_collections.my_namespace.my_collection.plugins.module_utils.ServiceAuth
# Create the module spec like normal
$spec
=
@{
options
=
@{
resource_id
=
@{
type
=
'str'
;
required
=
$true
}
state
=
@{
type
=
'str'
;
choices
=
'absent'
,
'present'
}
}
}
# Create the module from the module spec but also include the util spec to merge into our own.
$module
=
[Ansible.Basic.AnsibleModule]
::
Create
(
$args
,
$spec
,
@(
Get-MyNamespaceMyCollectionServiceAuthSpec
))
# Call the ServiceAuth module util and pass in the module object so it can access the module options.
Invoke-MyServiceResource
-Module
$module
-ResourceId
```

$module
.
Params
.
resource_id
-State
$module
.
params
.
state
$module
.
ExitJson
()
Note
Options defined in the module spec will always have precedence over a util spec. Any list values under the same key in a util spec will be appended to the module spec for that same key. Dictionary values will add any keys that are missing from the module spec and merge any values that are lists or dictionaries. This is similar to how the doc fragment plugins work when extending module documentation.
To document these shared util options for a module, create a doc fragment plugin that documents the options implemented
by the module util and extend the module docs for every module that implements the util to include that fragment in its docs.
Windows playbook module testing
?
You can test a module with an Ansible playbook. For example:
Create a playbook in any directory
touch
testmodule.yml
.
Create an inventory file in the same directory
touch
hosts
.
Populate the inventory file with the variables required to connect to a Windows host(s).
Add the following to the new playbook file:

-
name
:
test out windows module
hosts
:
windows
tasks
:
-
name
:
test out module
win_module
:
name

:
test name
Run the playbook
ansible-playbook
-i
hosts
testmodule.yml
This can be useful for seeing how Ansible runs with
the new module end to end. Other possible ways to test the module are
shown below.
Windows debugging
?
Debugging a module currently can only be done on a Windows host. This can be
useful when developing a new module or implementing bug fixes. These
are some steps that need to be followed to set this up:
Copy the module script to the Windows server
Copy the folders
./lib/ansible/module_utils/powershell
and
./lib/ansible/module_utils/csharp
to the same directory as the script above
Add an extra
#
to the start of any
#Requires
-Module
lines in the module code, this is only required for any lines starting with
#Requires
-Module
Add the following to the start of the module script that was copied to the server:
Set $ErrorActionPreference to what's set during Ansible execution
$ErrorActionPreference
=
"Stop"
Set the first argument as the path to a JSON file that contains the module args
$args
=
@(
"
$(
$pwd
.
Path
)
\args.json"
)
Or instead of an args file, set $complex_args to the pre-processed module args
$complex_args
=
@{
_ansible_check_mode
=
$false
_ansible_diff

=
$false
path
=
"C:\temp"
state
=
"present"
}
# Import any C# utils referenced with '#AnsibleRequires -CSharpUtil' or 'using Ansible.;
# The $_csharp_utils entries should be the context of the C# util files and not the path
Import-Module
-Name
"
$(
$pwd
.
Path
)
\powershell\Ansible.ModuleUtils.AddType.psm1"
$_csharp_utils
=
@(
[System.IO.File]
::
ReadAllText
(
"
$(
$pwd
.
Path
)
\csharp\Ansible.Basic.cs"
)
)
Add-CSharpType
-References
$_csharp_utils
-IncludeDebugInfo
Import any PowerShell modules referenced with '#Requires -Module`
Import-Module
-Name
"
$(
$pwd
.
Path
)
\powershell\Ansible.ModuleUtils.Legacy.psm1"
End of the setup code and start of the module code
#!powershell
You can add more args to
$complex_args

as required by the module or define the module options through a JSON file
with the structure:
{
"ANSIBLE_MODULE_ARGS"
:
{
"_ansible_check_mode"
:
false
,
"_ansible_diff"
:
false
,
"path"
:
"C:\\temp"
,
"state"
:
"present"
}
}
There are multiple IDEs that can be used to debug a Powershell script, two of
the most popular ones are
Powershell ISE
Visual Studio Code
To be able to view the arguments as passed by Ansible to the module follow
these steps.
Prefix the Ansible command with
ANSIBLE_KEEP_REMOTE_FILES=1
to specify that Ansible should keep the exec files on the server.
Log onto the Windows server using the same user account that Ansible used to execute the module.
Navigate to
%TEMP%\..
. It should contain a folder starting with
ansible-tmp-
.
Inside this folder, open the PowerShell script for the module.
In this script is a raw JSON script under
$json_raw
which contains the module arguments under
module_args
. These args can be assigned manually to the
$complex_args
variable that is defined on your debug script or put in the
args.json
file.
Windows unit testing
?
Currently there is no mechanism to run unit tests for Powershell modules under Ansible CI.
Windows integration testing
?
Integration tests for Ansible modules are typically written as Ansible roles. These test

roles are located in
./test/integration/targets
. You must first set up your testing
environment, and configure a test inventory for Ansible to connect to.
In this example we will set up a test inventory to connect to two hosts and run the integration
tests for win_stat:
Run the command
source
./hacking/env-setup
to prepare environment.
Create a copy of
./test/integration/inventory.winrm.template
and name it
inventory.winrm
.
Fill in entries under
[windows]
and set the required variables that are needed to connect to the host.
Install the required Python modules
to support WinRM and a configured authentication method.
To execute the integration tests, run
ansible-test
windows-integration
win_stat
; you can replace
win_stat
with the role you want to test.
This will execute all the tests currently defined for that role. You can set
the verbosity level using the
-v
argument just as you would with
ansible-playbook.
When developing tests for a new module, it is recommended to test a scenario once in
check mode and twice not in check mode. This ensures that check mode
does not make any changes but reports a change, as well as that the second run is
idempotent and does not report changes. For example:
-
name
:
remove a file (check mode)
win_file
:
path
:
C:\temp
state
:
absent
register
:
remove_file_check
check_mode
:
true

-
name
:
get result of remove a file (check mode)
win_command
:
powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }"
register
:
remove_file_actual_check
-
name
:
assert remove a file (check mode)
assert
:
that
:
-
remove_file_check is changed
-
remove_file_actual_check.stdout == 'true\r\n'
-
name
:
remove a file
win_file
:
path
:
C:\temp
state
:
absent
register
:
remove_file
-
name
:
get result of remove a file
win_command
:
powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }"
register
:
remove_file_actual
-
name
:
assert remove a file
assert
:
that

:
-
remove_file is changed
-
remove_file_actual.stdout == 'false\r\n'
-
name
:
remove a file (idempotent)
win_file
:
path
:
C:\temp
state
:
absent
register
:
remove_file_again
-
name
:
assert remove a file (idempotent)
assert
:
that
:
-
not remove_file_again is changed
Windows communication and development support
?
Join the
Ansible Forum
and use the
windows
tag for discussions about Ansible development for Windows.
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_plugins.html

Developing plugins ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing plugins
Edit on GitHub
Developing plugins
?
Writing plugins in Python
Raising errors
String encoding
Plugin configuration & documentation standards
General precedence rules
Accessing configuration settings
Developing particular plugin types
Action plugins
Cache plugins
Callback plugins
Connection plugins
Filter plugins
Inventory plugins
Lookup plugins
Test plugins
Vars plugins
Plugins augment Ansible?s core functionality with logic and features that are accessible to all modules. Ansible collections include a number of handy plugins, and you can easily write your own. All plugins must:
be written in Python
raise errors
return strings in unicode
conform to Ansible?s configuration and documentation standards
Once you?ve reviewed these general guidelines, you can skip to the particular type of plugin you want to develop.
Writing plugins in Python
?
You must write your plugin in Python so it can be loaded by the
PluginLoader
and returned as a Python object that any module can use. Since your plugin will execute on the control node, you must write it in a
compatible version of Python
.
Raising errors
?
You should return errors encountered during plugin execution by raising

AnsibleError()
or a similar class with a message describing the error. When wrapping other exceptions into error messages, you should always use the
to_native
Ansible function to ensure proper string compatibility across Python versions:
from
ansible.module_utils.common.text.converters
import
to_native
try
:
cause_an_exception
()
except
Exception
as
e
:
raise
AnsibleError
(
'Something happened, this was original exception:
%s
'
%
to_native
(
e
))
Since Ansible evaluates variables only when they are needed, filter and test plugins should propagate the exceptions
jinja2.exceptions.UndefinedError
and
AnsibleUndefinedVariable
to ensure undefined variables are only fatal when necessary.
Check the different
AnsibleError objects
and see which one applies best to your situation.
Check the section on the specific plugin type you?re developing for type-specific error handling details.
String encoding
?
You must convert any strings returned by your plugin into Python?s unicode type. Converting to unicode ensures that these strings can run through Jinja2. To convert strings:
from
ansible.module_utils.common.text.converters
import
to_text
result_string
=
to_text
(
result_string
)
Plugin configuration & documentation standards
?

To define configurable options for your plugin, describe them in the
DOCUMENTATION
section of the python file. Callback and connection plugins have declared configuration requirements this way since Ansible version 2.4; most plugin types now do the same. This approach ensures that the documentation of your plugin?s options will always be correct and up-to-date. To add a configurable option to your plugin, define it in this format:

```
options
:
option_name
:
description
:
describe this config option
default
:
default value for this config option
env
:
-
name
:
MYCOLLECTION_NAME_ENV_VAR_NAME
ini
:
-
section
:
mycollection_section_of_ansible.cfg_where_this_config_option_is_defined
key
:
key_used_in_ansible.cfg
vars
:
-
name
:
mycollection_name_of_ansible_var
-
name
:
mycollection_name_of_second_var
version_added
:
X.x
required
:
True/False
type
:
boolean/float/integer/list/none/path/pathlist/pathspec/string/tmppath
version_added
:
X.x
```

The supported configuration fields are:
env

List of environment variables that can be used to set this option.
Each entry includes a
name
field specifying the environment variable name.
The name should be in uppercase and should be prefixed with the collection name.
Multiple environment variables can be listed for the same option.
The last set environment variable in the list takes precedence if multiple are set.
This is commonly used for plugins (especially inventory plugins) to allow configuration through environment variables.
Examples:
VMWARE_PORT
,
GRAFANA_PASSWORD
ini
List of configuration file settings that can be used to set this option.
Each entry includes a
section
field for the configuration file section and a
key
field for the configuration key. Both should be in lowercase and should be prefixed with the collection name.
Multiple configuration settings can be listed for the same option.
The last set configuration setting in the list takes precedence if multiple are set.
This allows plugins to be configured with ansible.cfg.
Example:
grafana_password
vars
List of Ansible variables that can be used to set this option.
Each entry includes a
name
field specifying the variable name.
The name should be in lowercase and should be prefixed with the collection name.
Multiple variables can be listed for the same option.
The last set variable in the list takes precedence if multiple are set.
Variables follow Ansible?s variable precedence rules.
This allows plugins to be configured with Ansible variables.
Example:
ansible_vmware_port
General precedence rules
?
The precedence rules for configuration sources are listed below, starting with the highest precedence values:
Keywords
CLI settings
Environment variables (
env
)
Values defined in
ansible.cfg
Default value for the option, if present.
Accessing configuration settings
?
To access the configuration settings in your plugin, use
self.get_option(<option_name>)
.
Some plugin types handle this differently:
Become, callback, connection and shell plugins are guaranteed to have the engine call

set_options()
.
Lookup plugins always require you to handle it in the
run()
method.
Inventory plugins are done automatically if you use the
base
_read_config_file()
method. If not, you must use
self.get_option(<option_name>)
.
Cache plugins do it on load.
Cliconf, httpapi and netconf plugins indirectly piggy back on connection plugins.
Vars plugin settings are populated when first accessed (using the
self.get_option()
or
self.get_options()
method.
If you need to populate settings explicitly, use a
self.set_options()
call.
Configuration sources follow the precedence rules for values in Ansible. When there are multiple values from the same category, the value defined last takes precedence. For example, in the above configuration block, if both
name_of_ansible_var
and
name_of_second_var
are defined, the value of the
option_name
option will be the value of
name_of_second_var
. Refer to
Controlling how Ansible behaves: precedence rules
for further information.
Plugins that support embedded documentation (see
ansible-doc
for the list) should include well-formed doc strings. If you inherit from a plugin, you must document the options it takes, either through a documentation fragment or as a copy. See
Module format and documentation
for more information on correct documentation. Thorough documentation is a good idea even if you?re developing a plugin for local use.
In ansible-core 2.14 we added support for documenting filter and test plugins. You have two options for providing documentation:
Define a Python file that includes inline documentation for each plugin.
Define a Python file for multiple plugins and create adjacent documentation files in YAML format.
Developing particular plugin types
?
Action plugins
?
Action plugins let you integrate local processing and local data with module functionality.
To create an action plugin, create a new class with the Base(ActionBase) class as the parent:
from
ansible.plugins.action
import
ActionBase

class
ActionModule
(
ActionBase
):
pass
From there, execute the module using the
_execute_module
method to call the original module.
After successful execution of the module, you can modify the module return data.
module_return
=
self
.
_execute_module
(
module_name
=
'<NAME_OF_MODULE>'
,
module_args
=
module_args
,
task_vars
=
task_vars
,
tmp
=
tmp
)
For example, if you wanted to check the time difference between your Ansible control node and your target machine(s), you could write an action plugin to check the local time and compare it to the return data from Ansible?s
setup
module:
#!/usr/bin/python
Make coding more python3-ish, this is required for contributions to Ansible
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
from
ansible.plugins.action
import

```
ActionBase
from
datetime
import
datetime
class
ActionModule
(
ActionBase
):
def
run
(
self
,
tmp
=
None
,
task_vars
=
None
):
super
(
ActionModule
,
self
)
.
run
(
tmp
,
task_vars
)
module_args
=
self
.
_task
.
args
.
copy
()
module_return
=
self
.
_execute_module
(
module_name
=
```

```
'setup'
,
module_args
=
module_args
,
task_vars
=
task_vars
,
tmp
=
tmp
)
ret
=
dict
()
remote_date
=
None
if
not
module_return
.
get
(
'failed'
):
for
key
,
value
in
module_return
[
'ansible_facts'
]
.
items
():
if
key
==
'ansible_date_time'
:
remote_date
=
value
[
'iso8601'
]
if
remote_date
```

```
:
remote_date_obj
=
datetime
.
strptime
(
remote_date
,
'%Y-%m-
%d
T%H:%M:%SZ'
)
time_delta
=
datetime
.
utcnow
()
-
remote_date_obj
ret
[
'delta_seconds'
]
=
time_delta
.
seconds
ret
[
'delta_days'
]
=
time_delta
.
days
ret
[
'delta_microseconds'
]
=
time_delta
.
microseconds
return
dict
(
ansible_facts
=
dict
(
ret
))
```

This code checks the time on the control node, captures the date and time for the remote machine using the
setup
module, and calculates the difference between the captured time and
the local time, returning the time delta in days, seconds and microseconds.
For practical examples of action plugins,
see the source code for the
action plugins included with Ansible Core
Cache plugins
?
Cache plugins store gathered facts and data retrieved by inventory plugins.
Import cache plugins using the cache_loader so you can use
self.set_options()
and
self.get_option(<option_name>)
. If you import a cache plugin directly in the code base, you can only access options by the
ansible.constants
, and you break the cache plugin?s ability to be used by an inventory plugin.
from
ansible.plugins.loader
import
cache_loader
[
...
]
plugin
=
cache_loader
.
get
(
'custom_cache'
,
**
cache_kwargs
)
There are two base classes for cache plugins,
BaseCacheModule
for database-backed caches, and
BaseCacheFileModule
for file-backed caches.
To create a cache plugin, start by creating a new
CacheModule
class with the appropriate base class. If you?re creating a plugin using an
__init__
method you should initialize the base class with any provided args and kwargs to be compatible with inventory plugin cache options. The base class calls
self.set_options(direct=kwargs)
. After the base class
__init__
method is called
self.get_option(<option_name>)
should be used to access cache options.
New cache plugins should take the options
_uri

,
_prefix
, and
_timeout
to be consistent with existing cache plugins.
from
ansible.plugins.cache
import
BaseCacheModule
class
CacheModule
(
BaseCacheModule
):
def
__init__
(
self
,
*
args
,
**
kwargs
):
super
(
CacheModule
,
self
)
.
__init__
(
*
args
,
**
kwargs
)
self
.
_connection
=
self
.
get_option
(
'_uri'
)
self
.
_prefix
=

self
.
get_option
(
'_prefix'
)
self
.
_timeout
=
self
.
get_option
(
'_timeout'
)
If you use the
BaseCacheModule
, you must implement the methods
get
,
contains
,
keys
,
set
,
delete
,
flush
, and
copy
. The
contains
method should return a boolean that indicates if the key exists and has not expired. Unlike file-based caches, the
get
method does not raise a KeyError if the cache has expired.
If you use the
BaseFileCacheModule
, you must implement
_load
and
_dump
methods that will be called from the base class methods
get
and
set
.
If your cache plugin stores JSON, use
AnsibleJSONEncoder
in the
_dump
or
set

method and
AnsibleJSONDecoder
in the
_load
or
get
method.
For example cache plugins, see the source code for the
cache plugins included with Ansible Core
.
Callback plugins
?
Callback plugins add new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line programs.
To create a callback plugin, create a new class with the Base(Callbacks) class as the parent:
from
ansible.plugins.callback
import
CallbackBase
class
CallbackModule
(
CallbackBase
):
pass
From there, override the specific methods from the CallbackBase that you want to provide a callback for.
For plugins intended for use with Ansible version 2.0 and later, you should only override methods that start with
v2
.
For a complete list of methods that you can override, please see
__init__.py
in the
lib/ansible/plugins/callback
directory.
The following is a modified example of how Ansible?s timer plugin is implemented,
but with an extra option so you can see how configuration works in Ansible version 2.4 and later:
Make coding more python3-ish, this is required for contributions to Ansible
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
not only visible to ansible-doc, it also 'declares' the options the plugin requires and how to configure them.
DOCUMENTATION
=
'''

name: timer
callback_type: aggregate
requirements:
- enable in configuration
short_description: Adds time to play stats
version_added: "2.0" # for collections, use the collection version, not the Ansible version
description:
- This callback just adds total play duration to the play stats.
options:
format_string:
description: format of the string shown to user at play end
ini:
- section: callback_timer
key: format_string
env:
- name: ANSIBLE_CALLBACK_TIMER_FORMAT
default: "Playbook run took
%s
days,
%s
hours,
%s
minutes,
%s
seconds"
'''
from
datetime
import
datetime
from
ansible.plugins.callback
import
CallbackBase
class
CallbackModule
(
CallbackBase
):
"""
This callback module tells you how long your plays ran for.
"""
CALLBACK_VERSION
=
2.0
CALLBACK_TYPE
=
'aggregate'
CALLBACK_NAME
=
'namespace.collection_name.timer'
only needed if you ship it and don't want to enable by default
CALLBACK_NEEDS_ENABLED
=

```
True
def
__init__
(
self
):
# make sure the expected objects are present, calling the base's __init__
super
(
CallbackModule
,
self
)
.
__init__
()
# start the timer when the plugin is loaded, the first play should start a few milliseconds after.
self
.
start_time
=
datetime
.
now
()
def
_days_hours_minutes_seconds
(
self
,
runtime
):
''' internal helper method for this callback '''
minutes
=
(
runtime
.
seconds
//
60
)
%
60
r_seconds
=
runtime
.
seconds
-
(
minutes
*
60
```

```
)
return
runtime
.
days
,
runtime
.
seconds
//
3600
,
minutes
,
r_seconds
# this is only event we care about for display, when the play shows its summary stats; the rest are ignored by the base class
def
v2_playbook_on_stats
(
self
,
stats
):
end_time
=
datetime
.
now
()
runtime
=
end_time
-
self
.
start_time
# Shows the usage of a config option declared in the DOCUMENTATION variable. Ansible will have set it when it loads the plugin.
# Also note the use of the display object to print to screen. This is available to all callbacks, and you should use this over printing yourself
self
.
_display
.
display
(
self
.
_plugin_options
[
'format_string'
]
%
```

(
self
.
_days_hours_minutes_seconds
(
runtime
)))
Note that the
CALLBACK_VERSION
and
CALLBACK_NAME
definitions are required for properly functioning plugins for Ansible version 2.0 and later.
CALLBACK_TYPE
is mostly needed to distinguish ?stdout? plugins from the rest, since you can only load one plugin that writes to stdout.
For example callback plugins, see the source code for the
callback plugins included with Ansible Core
New in ansible-core 2.11, callback plugins are notified (by the
v2_playbook_on_task_start
) of
meta
tasks. By default, only explicit
meta
tasks that users list in their plays are sent to callbacks.
There are also some tasks which are generated internally and implicitly at various points in execution. Callback plugins can opt-in to receiving these implicit tasks as well, by setting
self.wants_implicit_tasks
=
True
. Any
Task
object received by a callback hook will have an
.implicit
attribute, which can be consulted to determine whether the
Task
originated from within Ansible, or explicitly by the user.
Connection plugins
?
Connection plugins allow Ansible to connect to target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time. The most commonly used connection plugins are native
ssh
,
paramiko
, and
local
. All of these can be used with ad-hoc tasks and in playbooks.
To create a new connection plugin (for example, to support SNMP, Message bus, or other transports), copy the format of one of the existing connection plugins and drop it into
connection
directory on your
local plugin path
.
Connection plugins can support common options (such as the
--timeout
flag) by defining an entry in the documentation for the attribute name (in this case

timeout
). If the common option has a non-null default, the plugin should define the same default since a different default would be ignored.
For example connection plugins, see the source code for the
connection plugins included with Ansible Core
.
Filter plugins
?
Filter plugins manipulate data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the
template
module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the filter plugins shipped with Ansible reside in a
core.py
.
Filter plugins do not use the standard configuration system described above, but since ansible-core 2.14 can use it as plain documentation.
Since Ansible evaluates variables only when they are needed, filter plugins should propagate the exceptions
jinja2.exceptions.UndefinedError
and
AnsibleUndefinedVariable
to ensure undefined variables are only fatal when necessary.
try
:
cause_an_exception
(
with_undefined_variable
)
except
jinja2
.
exceptions
.
UndefinedError
as
e
:
raise
AnsibleUndefinedVariable
(
"Something happened, this was the original exception:
%s
"
%
to_native
(
e
))
except
Exception
as
e
:
raise
AnsibleFilterError

(
"Something happened, this was the original exception:
%s
"
%
to_native
(
e
))
For example filter plugins, see the source code for the
filter plugins included with Ansible Core
.
Inventory plugins
?
Inventory plugins parse inventory sources and form an in-memory representation of the inventory. Inventory plugins were added in Ansible version 2.4.
You can see the details for inventory plugins in the
Developing dynamic inventory
page.
Lookup plugins
?
Lookup plugins pull in data from external data stores. Lookup plugins can be used within playbooks both for looping ? playbook language constructs like
with_fileglob
and
with_items
are implemented through lookup plugins ? and to return values into a variable or parameter.
Lookup plugins are expected to return lists, even if just a single element.
Ansible includes many
filters
which can be used to manipulate the data returned by a lookup plugin. Sometimes it makes sense to do the filtering inside the lookup plugin, other times it is better to return results that can be filtered in the playbook. Keep in mind how the data will be referenced when determining the appropriate level of filtering to be done inside the lookup plugin.
Here?s a simple lookup plugin implementation ? this lookup returns the contents of a text file as a variable:
python 3 headers, required if submitting to Ansible
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
DOCUMENTATION
=
r
"""
name: file
author: Daniel Hokka Zakrisson (@dhozac) <

```
[email protected]
>
version_added: "0.9"  # for collections, use the collection version, not the Ansible version
short_description: read file contents
description:
- This lookup returns the contents from a file on the Ansible control node's file system.
options:
_terms:
description: path(s) of files to read
required: True
option1:
description:
- Sample option that could modify plugin behavior.
- This one can be set directly ``option1='x'`` or in ansible.cfg, but can also use vars or environment.
type: string
ini:
- section: file_lookup
key: option1
notes:
- if read in variable context, the file can be interpreted as YAML if the content is valid to the parser.
- this lookup does not understand globbing --- use the fileglob lookup instead.
"""
from
ansible.errors
import
AnsibleError
,
AnsibleParserError
from
ansible.plugins.lookup
import
LookupBase
from
ansible.utils.display
import
Display
display
=
Display
()
class
LookupModule
(
LookupBase
):
def
run
(
self
,
terms
,
variables
=
```

```
None
,
**
kwargs
):
# First of all populate options,
# this will already take into account env vars and ini config
self
.
set_options
(
var_options
=
variables
,
direct
=
kwargs
)
# lookups in general are expected to both take a list as input and output a list
# this is done so they work with the looping construct 'with_'.
ret
=
[]
for
term
in
terms
:
display
.
debug
(
"File lookup term:
%s
"
%
term
)
# Find the file in the expected search path, using a class method
# that implements the 'expected' search path for Ansible plugins.
lookupfile
=
self
.
find_file_in_search_path
(
variables
,
'files'
,
term
)
# Don't use print or your own logging, the display class
```

```
# takes care of it in a unified way.
display
.
vvvv
(
u
"File lookup using
%s
as file"
%
lookupfile
)
try
:
if
lookupfile
:
contents
,
show_data
=
self
.
_loader
.
_get_file_contents
(
lookupfile
)
ret
.
append
(
contents
.
rstrip
())
else
:
# Always use ansible error classes to throw 'final' exceptions,
# so the Ansible engine will know how to deal with them.
# The Parser error indicates invalid options passed
raise
AnsibleParserError
()
except
AnsibleParserError
:
raise
AnsibleError
(
"could not locate file in lookup:
%s
"
```

%
term
)
consume an option: if this did something useful, you can retrieve the option value here
if
self
.
get_option
(
'option1'
)
==
'do something'
:
pass
return
ret
The following is an example of how this lookup is called:

-
hosts
:
all
vars
:
contents
:
"{{
lookup('namespace.collection_name.file',
'/etc/foo.txt')
}}"
contents_with_option
:
"{{
lookup('namespace.collection_name.file',
'/etc/foo.txt',
option1='donothing')
}}"
tasks
:
-
debug
:
msg
:
the value of foo.txt is {{ contents }} as seen today {{ lookup('pipe', 'date +"%Y-%m-%d"') }}
For example lookup plugins, see the source code for the
lookup plugins included with Ansible Core
.
For more usage examples of lookup plugins, see
Using Lookups
.
Test plugins
?

Test plugins verify data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the
template
module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the test plugins shipped with Ansible reside in a
core.py
. These are specially useful in conjunction with some filter plugins like
map
and
select
; they are also available for conditional directives like
when:
.
Test plugins do not use the standard configuration system described above. Since ansible-core 2.14 test plugins can use plain documentation.
Since Ansible evaluates variables only when they are needed, test plugins should propagate the exceptions
jinja2.exceptions.UndefinedError
and
AnsibleUndefinedVariable
to ensure undefined variables are only fatal when necessary.
try
:
cause_an_exception
(
with_undefined_variable
)
except
jinja2
.
exceptions
.
UndefinedError
as
e
:
raise
AnsibleUndefinedVariable
(
"Something happened, this was the original exception:
%s
"
%
to_native
(
e
))
except
Exception
as
e
:
raise
AnsibleFilterError
(
"Something happened, this was the original exception:

%s
"
%
to_native
(
e
))
For example test plugins, see the source code for the
test plugins included with Ansible Core
.
Vars plugins
?
Vars plugins inject additional variable data into Ansible runs that did not come from an inventory source, playbook, or command line. Playbook constructs like ?host_vars? and ?group_vars? work using vars plugins.
Vars plugins were partially implemented in Ansible 2.0 and rewritten to be fully implemented starting with Ansible 2.4. Vars plugins are supported by collections starting with Ansible 2.10.
Older plugins used a
run
method as their main body/work:
def
run
(
self
,
name
,
vault_password
=
None
):
pass
your code goes here
Ansible 2.0 did not pass passwords to older plugins, so vaults were unavailable.
Most of the work now happens in the
get_vars
method which is called from the VariableManager when needed.
def
get_vars
(
self
,
loader
,
path
,
entities
):
pass
your code goes here
The parameters are:
loader: Ansible?s DataLoader. The DataLoader can read files, auto-load JSON/YAML and decrypt vaulted data, and cache read files.
path: this is ?directory data? for every inventory source and the current play?s playbook directory, so they can search for data in reference to them.

get_vars
will be called at least once per available path.
entities: these are host or group names that are pertinent to the variables needed. The plugin will get called once for hosts and again for groups.
This
get_vars
method just needs to return a dictionary structure with the variables.
Since Ansible version 2.4, vars plugins only execute as needed when preparing to execute a task. This avoids the costly ?always execute? behavior that occurred during inventory construction in older versions of Ansible. Since Ansible version 2.10, vars plugin execution can be toggled by the user to run when preparing to execute a task or after importing an inventory source.
The user must explicitly enable vars plugins that reside in a collection. See
Enabling vars plugins
for details.
Legacy vars plugins are always loaded and run by default. You can prevent them from automatically running by setting
REQUIRES_ENABLED
to True.
class
VarsModule
(
BaseVarsPlugin
):
REQUIRES_ENABLED
=
True
Include the
vars_plugin_staging
documentation fragment to allow users to determine when vars plugins run.
DOCUMENTATION
=
'''
name: custom_hostvars
version_added: "2.10" # for collections, use the collection version, not the Ansible version
short_description: Load custom host vars
description: Load custom host vars
options:
stage:
ini:
- key: stage
section: vars_custom_hostvars
env:
- name: ANSIBLE_VARS_PLUGIN_STAGE
extends_documentation_fragment:
- vars_plugin_staging
'''
At times a value provided by a vars plugin will contain unsafe values. The utility function
wrap_var
provided by
ansible.utils.unsafe_proxy
should be used to ensure that Ansible handles the variable and value correctly. The use cases for unsafe data is covered in
Unsafe or raw strings
.
from

```
ansible.plugins.vars
import
BaseVarsPlugin
from
ansible.utils.unsafe_proxy
import
wrap_var
class
VarsPlugin
(
BaseVarsPlugin
):
def
get_vars
(
self
,
loader
,
path
,
entities
):
return
dict
(
something_unsafe
=
wrap_var
(
"{{ SOMETHING_UNSAFE }}"
)
)
```

For example vars plugins, see the source code for the
vars plugins included with Ansible Core
.

See also

Collection Index
Browse existing collections, modules, and plugins
Python API
Learn about the Python API for task execution
Developing dynamic inventory
Learn about how to develop dynamic inventory sources
Developing modules
Learn about how to write Ansible modules
Communication
Got questions? Need help? Want to share your ideas? Visit the Ansible communication guide
Adjacent YAML documentation files
Alternate YAML files as documentation
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_program_flow_modules.html

Ansible module architecture ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Developing
ansible-core
Ansible module architecture
Edit on GitHub
Ansible module architecture
?
If you are working on the
ansible-core
code, writing an Ansible module, or developing an action plugin, you may need to understand how Ansible?s program flow executes. If you are just using Ansible Modules in playbooks, you can skip this section.
Types of modules
Action plugins
New-style modules
Python
PowerShell
JSONARGS modules
Non-native want JSON modules
Binary modules
Old-style modules
How modules are executed
Executor/task_executor
The
normal
action plugin
Executor/module_common.py
Assembler frameworks
Module Replacer framework
Ansiballz framework
Passing args
Internal arguments
_ansible_no_log
_ansible_debug
_ansible_diff
_ansible_verbosity
_ansible_selinux_special_fs
_ansible_syslog_facility
_ansible_version
_ansible_module_name

_ansible_keep_remote_files
_ansible_socket
_ansible_shell_executable
_ansible_tmpdir
_ansible_remote_tmp
Module return values & Unsafe strings
Special considerations
Pipelining
Why pass args over stdin?
AnsibleModule
Argument spec
Dependencies between module options
Declaring check mode support
Adding file options
Types of modules
?
Ansible supports several different types of modules in its code base. Some of
these are for backwards compatibility and others are to enable flexibility.
Action plugins
?
Action plugins look like modules to anyone writing a playbook. Usage documentation for most action plugins lives inside a module of the same name. Some action plugins do all the work, with the module providing only documentation. Some action plugins execute modules. The
normal
action plugin executes modules that don?t have special action plugins. Action plugins always execute on the control node.
Some action plugins do all their work on the control node. For
example, the
debug
action plugin (which prints text for
the user to see) and the
assert
action plugin (which
tests whether values in a playbook satisfy certain criteria) execute entirely on the control node.
Most action plugins set up some values on the control node, then invoke an
actual module on the managed node that does something with these values. For example, the
template
action plugin takes values from
the user to construct a file in a temporary location on the control node using
variables from the playbook environment. It then transfers the temporary file
to a temporary file on the remote system. After that, it invokes the
copy module
which operates on the remote system to move the file
into its final location, sets file permissions, and so on.
New-style modules
?
All of the modules that ship with Ansible fall into this category. While you can write modules in any language, all official modules (shipped with Ansible) use either Python or PowerShell.
New-style modules have the arguments to the module embedded inside of them in
some manner. Old-style modules must copy a separate file over to the
managed node, which is less efficient as it requires two over-the-wire
connections instead of only one.
Python
?

New-style Python modules use the
Ansiballz framework
framework for constructing
modules. These modules use imports from
ansible.module_utils
to pull in
boilerplate module code, such as argument parsing, formatting of return
values as
JSON
, and various file operations.
Note
In Ansible, up to version 2.0.x, the official Python modules used the
Module Replacer framework
framework. For module authors,
Ansiballz framework
is
largely a superset of
Module Replacer framework
functionality, so you usually
do not need to understand the differences between them.
PowerShell
?
New-style PowerShell modules use the
Module Replacer framework
framework for
constructing modules. These modules get a library of PowerShell code embedded
in them before being sent to the managed node.
JSONARGS modules
?
These modules are scripts that include the string
<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>
in their body.
This string is replaced with the JSON-formatted argument string. These modules typically set a variable to that value like this:
json_arguments
=
"""<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>"""
Which is expanded as:
json_arguments
=
"""{"param1": "test's quotes", "param2": "
\"
To be or not to be
\"
- Hamlet"}"""
Note
Ansible outputs a
JSON
string with bare quotes. Double quotes are
used to quote string values, double quotes inside of string values are
backslash escaped, and single quotes may appear unescaped inside of
a string value. To use JSONARGS, your scripting language must have a way
to handle this type of string. The example uses Python?s triple quoted
strings to do this. Other scripting languages may have a similar quote

character that won?t be confused by any quotes in the JSON or it may allow you to define your own start-of-quote and end-of-quote characters. If the language doesn?t give you any of these then you?ll need to write a non-native JSON module or Old-style module instead. These modules typically parse the contents of json_arguments using a JSON library and then use them as native variables throughout the code.

Non-native want JSON modules?

If a module has the string WANT_JSON in it anywhere, Ansible treats it as a non-native module that accepts a file name as its only command-line parameter. The file name is for a temporary file containing a JSON string containing the module?s parameters. The module needs to open the file, read and parse the parameters, operate on the data, and print its return data as a JSON encoded dictionary to stdout before exiting.

These types of modules are self-contained entities. As of Ansible 2.1, Ansible only modifies them to change a shebang line if present.

See also

Examples of Non-native modules written in ruby are in the Ansible for Rubyists repository.

Binary modules?

From Ansible 2.2 onwards, modules may also be small binary programs. Ansible doesn?t perform any magic to make these portable to different systems so they may be specific to the system on which they were compiled or require other binary runtime dependencies. Despite these drawbacks, you may have to compile a custom module against a specific binary library if that?s the only way to get access to certain resources.

Binary modules take their arguments and return data to Ansible in the same way as want JSON modules.

See also

One example of a binary module written in go.

Old-style modules?

Old-style modules are similar to want JSON modules, except that the file that they take contains key=value pairs for their parameters instead of

JSON
. Ansible decides that a module is old-style when it doesn?t have
any of the markers that would show that it is one of the other types.
How modules are executed
?
When a user uses
ansible
or
ansible-playbook
, they
specify a task to execute. The task is usually the name of a module along
with several parameters to be passed to the module. Ansible takes these
values and processes them in various ways before they are finally executed on
the remote machine.
Executor/task_executor
?
The TaskExecutor receives the module name and parameters that were parsed from
the
playbook
(or from the command-line in the case of
/usr/bin/ansible
). It uses the name to decide whether it is looking
at a module or an
Action Plugin
. If it is
a module, it loads the
Normal Action Plugin
and passes the name, variables, and other information about the task and play
to that Action Plugin for further processing.
The
normal
action plugin
?
The
normal
action plugin executes the module on the remote host. It is
the primary coordinator of much of the work to actually execute the module on
the managed machine.
It loads the appropriate connection plugin for the task, which then transfers
or executes as needed to create a connection to that host.
It adds any internal Ansible properties to the module?s parameters (for
instance, the ones that pass along
no_log
to the module).
It works with other plugins (connection, shell, become, other action plugins)
to create any temporary files on the remote machine and
cleans up afterwards.
It pushes the module and module parameters to the
remote host, although the
module_common
code described in the next section decides which format
those will take.
It handles any special cases regarding modules (for example, async
execution, or complications around Windows modules that must have the same names as Python modules, so that

internal calling of modules from other Action Plugins work.)
Much of this functionality comes from the
BaseAction
class,
which lives in
plugins/action/__init__.py
. It uses the
Connection
and
Shell
objects to do its work.
Note
When
tasks
are run with the
async:
parameter, Ansible
uses the
async
Action Plugin instead of the
normal
Action Plugin
to invoke it. That program flow is currently not documented. Read the
source for information on how that works.
Executor/module_common.py
?
Code in
executor/module_common.py
assembles the module
to be shipped to the managed node. The module is first read in, then examined
to determine its type:
PowerShell
and
JSON-args modules
are passed through
Module Replacer
.
New-style
Python modules
are assembled by
Ansiballz framework
.
Non-native-want-JSON
,
Binary modules
, and
Old-Style modules
aren?t touched by either of these and pass through unchanged.
After the assembling step, one final
modification is made to all modules that have a shebang line. Ansible checks
whether the interpreter in the shebang line has a specific path configured with
an
ansible_$X_interpreter
inventory variable. If it does, Ansible

substitutes that path for the interpreter path given in the module. After
this, Ansible returns the complete module data and the module type to the
Normal Action
which continues execution of
the module.
Assembler frameworks
?
Ansible supports two assembler frameworks: Ansiballz and the older Module Replacer.
Module Replacer framework
?
The Module Replacer framework is the original framework implementing new-style
modules, and is still used for PowerShell modules. It is essentially a preprocessor (like the C Preprocessor for those familiar with that programming language). It does straight substitutions of
specific substring patterns in the module file. There are two types of
substitutions:
Replacements that only happen in the module file. These are public
replacement strings that modules can utilize to get helpful boilerplate or
access to arguments.
from
ansible.module_utils.MOD_LIB_NAME
import
*
is replaced with the
contents of the
ansible/module_utils/MOD_LIB_NAME.py
These should
only be used with
new-style Python modules
.
#<<INCLUDE_ANSIBLE_MODULE_COMMON>>
is equivalent to
from
ansible.module_utils.basic
import
*
and should also only apply
to new-style Python modules.
#
POWERSHELL_COMMON
substitutes the contents of
ansible/module_utils/powershell.ps1
. It should only be used with
new-style Powershell modules
.
Replacements that are used by
ansible.module_utils
code. These are internal replacement patterns. They may be used internally, in the above public replacements, but shouldn?t be used directly by modules.
"<<ANSIBLE_VERSION>>"
is substituted with the Ansible version. In
new-style Python modules
under the
Ansiballz framework
framework the proper way is to instead instantiate an

AnsibleModule
and then access the version from
:attr:
AnsibleModule.ansible_version
.
"<<INCLUDE_ANSIBLE_MODULE_COMPLEX_ARGS>>"
is substituted with
a string which is the Python
repr
of the
JSON
encoded module
parameters. Using
repr
on the JSON string makes it safe to embed in
a Python file. In new-style Python modules under the Ansiballz framework
this is better accessed by instantiating an
AnsibleModule
and
then using
AnsibleModule.params
.
<<SELINUX_SPECIAL_FILESYSTEMS>>
substitutes a string which is
a comma-separated list of file systems which have a file system dependent
security context in SELinux. In new-style Python modules, if you really
need this you should instantiate an
AnsibleModule
and then use
AnsibleModule._selinux_special_fs
. The variable has also changed
from a comma-separated string of file system names to an actual python
list of file system names.
<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>
substitutes the module
parameters as a JSON string. Care must be taken to properly quote the
string as JSON data may contain quotes. This pattern is not substituted
in new-style Python modules as they can get the module parameters another
way.
The string
syslog.LOG_USER
is replaced wherever it occurs with the
syslog_facility
which was named in
ansible.cfg
or any
ansible_syslog_facility
inventory variable that applies to this host. In
new-style Python modules this has changed slightly. If you really need to
access it, you should instantiate an
AnsibleModule
and then use
AnsibleModule._syslog_facility
to access it. It is no longer the

actual syslog facility and is now the name of the syslog facility. See
the
documentation on internal arguments
for details.
Ansiballz framework
?
The Ansiballz framework was adopted in Ansible 2.1 and is used for all new-style Python modules. Unlike the Module Replacer, Ansiballz uses real Python imports of things in
ansible/module_utils
instead of merely preprocessing the module. It
does this by constructing a zipfile ? which includes the module file, files
in
ansible/module_utils
that are imported by the module, and some
boilerplate to pass in the module?s parameters. The zipfile is then Base64
encoded and wrapped in a small Python script which decodes the Base64 encoding
and places the zipfile into a temp directory on the managed node. It then
extracts just the Ansible module script from the zip file and places that in
the temporary directory as well. Then it sets the PYTHONPATH to find Python
modules inside of the zip file and imports the Ansible module as the special name,
__main__
.
Importing it as
__main__
causes Python to think that it is executing a script rather than simply
importing a module. This lets Ansible run both the wrapper script and the module code in a single copy of Python on the remote machine.
Note
Ansible wraps the zipfile in the Python script for two reasons:
for compatibility with Python 2.6 which has a less
functional version of Python?s
-m
command-line switch.
so that pipelining will function properly. Pipelining needs to pipe the
Python module into the Python interpreter on the remote node. Python
understands scripts on stdin but does not understand zip files.
Prior to Ansible 2.7, the module was executed by a second Python interpreter instead of being
executed inside of the same process. This change was made once Python-2.4 support was dropped
to speed up module execution.
In Ansiballz, any imports of Python modules from the
ansible.module_utils
package trigger inclusion of that Python file
into the zipfile. Instances of
#<<INCLUDE_ANSIBLE_MODULE_COMMON>>
in
the module are turned into
from
ansible.module_utils.basic
import
*
and
ansible/module-utils/basic.py
is then included in the zipfile.
Files that are included from

module_utils
are themselves scanned for
imports of other Python modules from
module_utils
to be included in
the zipfile as well.
Passing args
?
Arguments are passed differently by the two frameworks:
In
Module Replacer framework
, module arguments are turned into a JSON-ified string and substituted into the combined module file.
In
Ansiballz framework
, the JSON-ified string is part of the script which wraps the zipfile. Just before the wrapper script imports the Ansible module as
__main__
, it monkey-patches the private,
_ANSIBLE_ARGS
variable in
basic.py
with the variable values. When a
ansible.module_utils.basic.AnsibleModule
is instantiated, it parses this string and places the args into
AnsibleModule.params
where it can be accessed by the module?s other code.
Warning
If you are writing modules, remember that the way we pass arguments is an internal implementation detail: it has changed in the past and will change again as soon as changes to the common module_utils
code allow Ansible modules to forgo using
ansible.module_utils.basic.AnsibleModule
. Do not rely on the internal global
_ANSIBLE_ARGS
variable.
Very dynamic custom modules which need to parse arguments before they
instantiate an
AnsibleModule
may use
_load_params
to retrieve those parameters.
Although
_load_params
may change in breaking ways if necessary to support
changes in the code, it is likely to be more stable than either the way we pass parameters or the internal global variable.
Note
Prior to Ansible 2.7, the Ansible module was invoked in a second Python interpreter and the
arguments were then passed to the script over the script?s stdin.
Internal arguments
?
Both
Module Replacer framework
and
Ansiballz framework
send additional arguments to

the Ansible module beyond those which the user specified in the playbook. These additional arguments are internal parameters that help implement global Ansible features. Modules often do not need to know about these explicitly because the features are implemented in ansible.module_utils.basic . However, certain features need support from modules and some knowledge of the internal arguments is useful.

The internal arguments in this section are global. If you need to add a local internal argument to a custom module, create an action plugin for that specific module. See _original_basename in the copy action plugin for an example.

_ansible_no_log ?

Type: bool

Set to True whenever an argument in a task or play specifies no_log . Any module that calls the AnsibleModule.log() function handles this action automatically. If you have a module that implements its own logging then you need to check the value of _ansible_no_log . To access _ansible_no_log in a module, instantiate the AnsibleModule utility and then check the value of AnsibleModule.no_log .

Note

no_log specified in a module?s argument_spec is handled by a different mechanism.

_ansible_debug ?

Type: bool

Operates verbose logging and logging of external commands that a module executes. If the module uses the AnsibleModule.debug() function rather than the AnsibleModule.log() function then the messages are only logged if you set the _ansible_debug argument to True . To access _ansible_debug in a module, instantiate the AnsibleModule

utility and access
AnsibleModule._debug
. For more details, see
DEFAULT_DEBUG
.
_ansible_diff
?
Type:
bool
With this parameter you can configure your module to show a unified diff of changes that will be applied to the templated files. To access
_ansible_diff
in a module, instantiate the
AnsibleModule
utility and access
AnsibleModule._diff
. You can also access this parameter using the
diff
keyword in your playbook, or the relevant environment variable. For more details, see
Playbook Keywords
and the
DIFF_ALWAYS
configuration option.
_ansible_verbosity
?
Type:
int
You can use this argument to control the level (0 for none) of verbosity in logging.
_ansible_selinux_special_fs
?
Type:
list
Elements:
strings
This argument provides modules with the names of file systems which should have a special SELinux context. They are used by the
AnsibleModule
methods which operate on files (changing attributes, moving, and copying).
Most modules can use the built-in
AnsibleModule
methods to manipulate files. To access in a module that needs to know about these special context file systems, instantiate
AnsibleModule
and examine the list in
AnsibleModule._selinux_special_fs
.
This argument replaces
ansible.module_utils.basic.SELINUX_SPECIAL_FS
from
Module Replacer framework
. In the module replacer framework the argument was formatted as a comma-separated string of file system names. Under the Ansiballz framework it is a list. You can access
_ansible_selinux_special_fs
using the corresponding environment variable. For more details, see the

DEFAULT_SELINUX_SPECIAL_FS
configuration option.
New in version 2.1.
_ansible_syslog_facility
?
This argument controls which syslog facility the module logs to. Most modules should just use the
AnsibleModule.log()
function which will then make use of this. If a module has to use this on its own, it should instantiate the
AnsibleModule
method and then retrieve the name of the syslog facility from
AnsibleModule._syslog_facility
. The Ansiballz code is less elegant than the
Module Replacer framework
code:
Old module_replacer way
import
syslog
syslog
.
openlog
(
NAME
,
0
,
syslog
.
LOG_USER
)
New Ansiballz way
import
syslog
facility_name
=
module
.
_syslog_facility
facility
=
getattr
(
syslog
,
facility_name
,
syslog
.
LOG_USER
)
syslog
.
openlog
(
NAME

,
0
,
facility
)
For more details, see the
DEFAULT_SYSLOG_FACILITY
configuration option.
New in version 2.1.
_ansible_version
?
This argument passes the version of Ansible to the module. To access it, a module should instantiate the
AnsibleModule
method and then retrieve the version from
AnsibleModule.ansible_version
. This replaces
ansible.module_utils.basic.ANSIBLE_VERSION
from
Module Replacer framework
.
New in version 2.1.
_ansible_module_name
?
Type:
str
This argument passes the information to modules about their name. For more details see, the configuration option
DEFAULT_MODULE_NAME
.
_ansible_keep_remote_files
?
Type:
bool
This argument provides instructions that modules must be ready if they need to keep the remote files. For more details, see the
DEFAULT_KEEP_REMOTE_FILES
configuration option.
_ansible_socket
?
This argument provides modules with a socket for persistent connections. The argument is created using the
PERSISTENT_CONTROL_PATH_DIR
configuration option.
_ansible_shell_executable
?
Type:
bool
This argument ensures that modules use the designated shell executable. For more details, see the
ansible_shell_executable
remote host environment parameter.
_ansible_tmpdir
?
Type:
str
This argument provides instructions to modules that all commands must use the designated temporary directory, if created. The action plugin designs this temporary directory.

Modules can access this parameter by using the public
tmpdir
property. The
tmpdir
property will create a temporary directory if the action plugin did not set the parameter.
The directory name is generated randomly, and the the root of the directory is determined by one of these:
DEFAULT_LOCAL_TMP
remote_tmp
system_tmpdirs
As a result, using the
ansible.cfg
configuration file to activate or customize this setting will not guarantee that you control the full value.
_ansible_remote_tmp
?
The module?s
tmpdir
property creates a randomized directory name in this directory if the action plugin did not set
_ansible_tmpdir
. For more details, see the
remote_tmp
parameter of the shell plugin.
Module return values & Unsafe strings
?
At the end of a module?s execution, it formats the data that it wants to return as a JSON string and prints the string to its stdout. The normal action plugin receives the JSON string, parses it into a Python dictionary, and returns it to the executor.
If Ansible templated every string return value, it would be vulnerable to an attack from users with access to managed nodes. If an unscrupulous user disguised malicious code as Ansible return value strings, and if those strings were then templated on the control node, Ansible could execute arbitrary code. To prevent this scenario, Ansible marks all strings inside returned data as
Unsafe
, emitting any Jinja2 templates in the strings verbatim, not expanded by Jinja2.
Strings returned by invoking a module through
ActionPlugin._execute_module()
are automatically marked as
Unsafe
by the normal action plugin. If another action plugin retrieves information from a module through some other means, it must mark its return data as
Unsafe
on its own.
In case a poorly-coded action plugin fails to mark its results as ?Unsafe,? Ansible audits the results again when they are returned to the executor,
marking all strings as
Unsafe
. The normal action plugin protects itself and any other code that it calls with the result data as a parameter. The check inside the executor protects the output of all other action plugins, ensuring that subsequent tasks run by Ansible will not template anything from those results either.
Special considerations
?
Pipelining
?
Ansible can transfer a module to a remote machine in one of two ways:
it can write out the module to a temporary file on the remote host and then
use a second connection to the remote host to execute it with the

interpreter that the module needs
or it can use what?s known as pipelining to execute the module by piping it
into the remote interpreter?s stdin.
Pipelining only works with modules written in Python at this time because
Ansible only knows that Python supports this mode of operation. Supporting
pipelining means that whatever format the module payload takes before being
sent over the wire must be executable by Python through stdin.
Why pass args over stdin?
?
Passing arguments through stdin was chosen for the following reasons:
When combined with
ANSIBLE_PIPELINING
, this keeps the module?s arguments from
temporarily being saved onto disk on the remote machine. This makes it
harder (but not impossible) for a malicious user on the remote machine to
steal any sensitive information that may be present in the arguments.
Command line arguments would be insecure as most systems allow unprivileged
users to read the full commandline of a process.
Environment variables are usually more secure than the commandline but some
systems limit the total size of the environment. This could lead to
truncation of the parameters if we hit that limit.
AnsibleModule
?
Argument spec
?
The
argument_spec
provided to
AnsibleModule
defines the supported arguments for a module, as well as their type, defaults and more.
Example
argument_spec
:
module
=
AnsibleModule
(
argument_spec
=
dict
(
top_level
=
dict
(
type
=
'dict'
,
options
=
dict
(
second_level

=
dict
(
default
=
True
,
type
=
'bool'
,
)
)
)
))
This section will discuss the behavioral attributes for arguments:
type
:
type
allows you to define the type of the value accepted for the argument. The default value for
type
is
str
. Possible values are:
str
list
dict
bool
int
float
path
raw
jsonarg
json
bytes
bits
The
raw
type, performs no type validation or type casting, and maintains the type of the passed value.
elements
:
elements
works in combination with
type
when
type='list'
.
elements
can then be defined as
elements='int'
or any other type, indicating that each element of the specified list should be of that type.
default
:
The

default
option allows sets a default value for the argument for the scenario when the argument is not provided to the module. When not specified, the default value is
None
.
fallback
:
fallback
accepts a
tuple
where the first argument is a callable (function) that will be used to perform the lookup, based on the second argument. The second argument is a list of values to be accepted by the callable.
The most common callable used is
env_fallback
which will allow an argument to optionally use an environment variable when the argument is not supplied.
Example:
username
=
dict
(
fallback
=
(
env_fallback
,
[
'ANSIBLE_NET_USERNAME'
]))
choices
:
choices
accepts a list of choices that the argument will accept. The types of
choices
should match the
type
.
required
:
required
accepts a boolean, either
True
or
False
that indicates that the argument is required. When not specified,
required
defaults to
False
. This should not be used in combination with
default
.
no_log
:
no_log
accepts a boolean, either

True
or
False
, that indicates explicitly whether or not the argument value should be masked in logs and output.
Note
In the absence of
no_log
, if the parameter name appears to indicate that the argument value is a password or passphrase (such as ?admin_password?), a warning will be shown and the value will be masked in logs but
not
output. To disable the warning and masking for parameters that do not contain sensitive information, set
no_log
to
False
.
aliases
:
aliases
accepts a list of alternative argument names for the argument, such as the case where the argument is
name
but the module accepts
aliases=['pkg']
to allow
pkg
to be interchangeably with
name
. Use of aliases can make module interfaces confusing, so we recommend adding them only when necessary. If you are updating argument names to fix a typo or improve the interface, consider moving the old names to
deprecated_aliases
rather than keeping them around indefinitely.
options
:
options
implements the ability to create a sub-argument_spec, where the sub options of the top level argument are also validated using the attributes discussed in this section. The example at the top of this section demonstrates use of
options
.
type
or
elements
should be
dict
is this case.
apply_defaults
:
apply_defaults
works alongside
options
and allows the
default
of the sub-options to be applied even when the top-level argument is not supplied.
In the example of the
argument_spec
at the top of this section, it would allow

module.params['top_level']['second_level']
to be defined, even if the user does not provide
top_level
when calling the module.
removed_in_version
:
removed_in_version
indicates which version of ansible-core or a collection a deprecated argument will be removed in. Mutually exclusive with
removed_at_date
, and must be used with
removed_from_collection
.
Example:
option
=
{
'type'
:
'str'
,
'removed_in_version'
:
'2.0.0'
,
'removed_from_collection'
:
'testns.testcol'
,
},
removed_at_date
:
removed_at_date
indicates that a deprecated argument will be removed in a minor ansible-core release or major collection release after this date. Mutually exclusive with
removed_in_version
, and must be used with
removed_from_collection
.
Example:
option
=
{
'type'
:
'str'
,
'removed_at_date'
:
'2020-12-31'
,
'removed_from_collection'
:
'testns.testcol'
,

},
removed_from_collection
:
Specifies which collection (or ansible-core) deprecates this deprecated argument. Specify
ansible.builtin
for ansible-core, or the collection?s name (format
foo.bar
). Must be used with
removed_in_version
or
removed_at_date
.
deprecated_aliases
:
Deprecates aliases of this argument. Must contain a list or tuple of dictionaries having some the following keys:
name
:
The name of the alias to deprecate. (Required.)
version
:
The version of ansible-core or the collection this alias will be removed in. Either
version
or
date
must be specified.
date
:
The a date after which a minor release of ansible-core or a major collection release will no longer contain this alias.. Either
version
or
date
must be specified.
collection_name
:
Specifies which collection (or ansible-core) deprecates this deprecated alias. Specify
ansible.builtin
for ansible-core, or the collection?s name (format
foo.bar
). Must be used with
version
or
date
.
Examples:
option
=
{
'type'
:
'str'
,
'aliases'
:

[
'foo'
,
'bar'
],
'deprecated_aliases'
:
[
{
'name'
:
'foo'
,
'version'
:
'2.0.0'
,
'collection_name'
:
'testns.testcol'
,
},
{
'name'
:
'foo'
,
'date'
:
'2020-12-31'
,
'collection_name'
:
'testns.testcol'
,
},
],
},
mutually_exclusive
:
If
options
is specified,
mutually_exclusive
refers to the sub-options described in
options
and behaves as in
Dependencies between module options
.
required_together
:
If
options
is specified,

required_together
refers to the sub-options described in
options
and behaves as in
Dependencies between module options
.
required_one_of
:
If
options
is specified,
required_one_of
refers to the sub-options described in
options
and behaves as in
Dependencies between module options
.
required_if
:
If
options
is specified,
required_if
refers to the sub-options described in
options
and behaves as in
Dependencies between module options
.
required_by
:
If
options
is specified,
required_by
refers to the sub-options described in
options
and behaves as in
Dependencies between module options
.
context
:
New in version 2.17.
You can set the value of the
context
key to a dict of custom content. This allows you to provide additional context in the argument spec. The content provided is not validated or utilized by the core engine.
Example:
option
=
{
'type'
:
'str'
,

'context'
:
{
'disposition'
:
'/properties/apiType'
,
},
'choices'
:
[
'http'
,
'soap'
],
}
Dependencies between module options
?
The following are optional arguments for
AnsibleModule()
:
module
=
AnsibleModule
(
argument_spec
,
mutually_exclusive
=
[
(
'path'
,
'content'
),
],
required_one_of
=
[
(
'path'
,
'content'
),
],
)
mutually_exclusive
:
Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names which are mutually exclusive. If more than one options of a list are specified together, Ansible will fail the module with an error.
Example:
mutually_exclusive
=
[

(
'path'
,
'content'
),
(
'repository_url'
,
'repository_filename'
),
],
In this example, the options
path
and
content
must not specified at the same time. Also the options
repository_url
and
repository_filename
must not be specified at the same time. But specifying
path
and
repository_url
is accepted.
To ensure that precisely one of two (or more) options is specified, combine
mutually_exclusive
with
required_one_of
.
required_together
:
Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names which are must be specified together. If at least one of these options are specified, the other ones from the same sequence must all be present.
Example:
required_together
=
[
(
'file_path'
,
'file_hash'
),
],
In this example, if one of the options
file_path
or
file_hash
is specified, Ansible will fail the module with an error if the other one is not specified.
required_one_of
:
Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names from which at least one must be specified. If none one of these options are specified, Ansible will fail module execution.
Example:

required_one_of
=
[
(
'path'
,
'content'
),
],
In this example, at least one of
path
and
content
must be specified. If none are specified, execution will fail. Specifying both is explicitly allowed; to prevent this, combine
required_one_of
with
mutually_exclusive
.
required_if
:
Must be a sequence of sequences. Every inner sequence describes one conditional dependency. Every sequence must have three or four values. The first two values are the option?s name and the option?s value which describes the condition. The further elements of the sequence are only needed if the option of that name has precisely this value.
If you want that all options in a list of option names are specified if the condition is met, use one of the following forms:
(
'option_name'
,
option_value
,
(
'option_a'
,
'option_b'
,
...
)),
(
'option_name'
,
option_value
,
(
'option_a'
,
'option_b'
,
...
),
False
),
If you want that at least one option of a list of option names is specified if the condition is met, use the following form:
(
'option_name'
,

option_value
,
(
'option_a'
,
'option_b'
,
...
),
True
),
Example:
required_if
=
[
(
'state'
,
'present'
,
(
'path'
,
'content'
),
True
),
(
'force'
,
True
,
(
'force_reason'
,
'force_code'
)),
],
In this example, if the user specifies
state=present
, at least one of the options
path
and
content
must be supplied (or both). To make sure that precisely one can be specified, combine
required_if
with
mutually_exclusive
.
On the other hand, if
force
(a boolean parameter) is set to
true
,

yes
and so on, both
force_reason
and
force_code
must be specified.
required_by
:
Must be a dictionary mapping option names to sequences of option names. If the option name in a dictionary key is specified, the option names it maps to must all also be specified. Note that instead of a sequence of option names, you can also specify one single option name.
Example:
required_by
=
{
'force'
:
'force_reason'
,
'path'
:
(
'mode'
,
'owner'
,
'group'
),
},
In the example, if
force
is specified,
force_reason
must also be specified. Also, if
path
is specified, then three three options
mode
,
owner
and
group
also must be specified.
Declaring check mode support
?
To declare that a module supports check mode, supply
supports_check_mode=True
to the
AnsibleModule()
call:
module
=
AnsibleModule
(
argument_spec

,
supports_check_mode
=
True
)
The module can determine whether it is called in check mode by checking the boolean value
module.check_mode
. If it evaluates to
True
, the module must take care not to do any modification.
If
supports_check_mode=False
is specified, which is the default value, the module will exit in check mode with
skipped=True
and message
remote
module
(<insert
module
name
here>)
does
not
support
check
mode
.
Adding file options
?
To declare that a module should add support for all common file options, supply
add_file_common_args=True
to the
AnsibleModule()
call:
module
=
AnsibleModule
(
argument_spec
,
add_file_common_args
=
True
)
You can find
a list of all file options here
. It is recommended that you make your
DOCUMENTATION
extend the doc fragment
ansible.builtin.files
(see
Documentation fragments
) in this case, to make sure that all these fields are correctly documented.
The helper functions

module.load_file_common_arguments()
and
module.set_fs_attributes_if_different()
can be used to handle these arguments for you:
argument_spec
=
{
'path'
:
{
'type'
:
'str'
,
'required'
:
True
,
},
}
module
=
AnsibleModule
(
argument_spec
,
add_file_common_args
=
True
)
changed
=
False
TODO do something with module.params['path'], like update its contents
Ensure that module.params['path'] satisfies the file options supplied by the user
file_args
=
module
.
load_file_common_arguments
(
module
.
params
)
changed
=
module
.
set_fs_attributes_if_different
(
file_args
,
changed

```
)
module
.
exit_json
(
changed
=
changed
)
```

Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/developing_python_3.html

Ansible and Python 3 ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support

Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Developer Guide
Ansible and Python 3
Edit on GitHub
Ansible and Python 3
?
The
ansible-core
code runs Python 3 (for specific versions check
Control Node Requirements
Contributors to
ansible-core
and to Ansible Collections should be aware of the tips in this document so that they can write code that will run on the same versions of Python as the rest of Ansible.
Minimum version of Python 3.x and Python 2.x
Developing Ansible code that supports Python 2 and Python 3
Understanding strings in Python 2 and Python 3
Control node string strategy: the Unicode Sandwich
Unicode Sandwich common borders: places to convert bytes to text in control node code
Reading and writing to files
Filesystem interaction
Interacting with other programs
Module string strategy: Native String
Module_utils string strategy: hybrid
Tips, tricks, and idioms for Python 2/Python 3 compatibility
Use forward-compatibility boilerplate
Prefix byte strings with
b_
Import Ansible?s bundled Python
six
library
Handle exceptions with
as
Update octal numbers
String formatting for control node code
Use
str.format()
for Python 2.6 compatibility
Use percent format with byte strings
We do have some considerations depending on the types of Ansible code:

code on the control node - code that runs on the machine where you invoke
/usr/bin/ansible
, only needs to support the control node?s Python versions.
modules - the code which Ansible transmits to and invokes on the managed machine. Modules need to support the ?managed node? Python versions, with some exceptions.
shared
module_utils
code - the common code that is used by modules to perform tasks and sometimes used by code on the control node.
Shared
module_utils
code needs to support the same range of Python as the modules.
However, the three types of code do not use the same string strategy. If you?re developing a module or some
module_utils
code, be sure to read the section on string strategy carefully.
Minimum version of Python 3.x and Python 2.x
?
See
Control Node Requirements
and
Managed Node Requirements
for the
specific versions supported.
Your custom modules can support any version of Python (or other languages) you want, but the above are the requirements for the code contributed to the Ansible project.
Developing Ansible code that supports Python 2 and Python 3
?
The best place to start learning about writing code that supports both Python 2 and Python 3
is
Lennart Regebro?s book: Porting to Python 3
.
The book describes several strategies for porting to Python 3. The one we?re
using is
to support Python 2 and Python 3 from a single code base
Understanding strings in Python 2 and Python 3
?
Python 2 and Python 3 handle strings differently, so when you write code that supports Python 3
you must decide what string model to use. Strings can be an array of bytes (like in C) or
they can be an array of text. Text is what we think of as letters, digits,
numbers, other printable symbols, and a small number of unprintable ?symbols?
(control codes).
In Python 2, the two types for these (
str
for bytes and
unicode
for text) are often used interchangeably. When dealing only
with ASCII characters, the strings can be combined, compared, and converted
from one type to another automatically. When non-ASCII characters are
introduced, Python 2 starts throwing exceptions due to not knowing what encoding
the non-ASCII characters should be in.
Python 3 changes this behavior by making the separation between bytes (
bytes
)
and text (
str

) more strict. Python 3 will throw an exception when
trying to combine and compare the two types. The programmer has to explicitly
convert from one type to the other to mix values from each.
In Python 3 it is immediately apparent to the programmer when code is
mixing the byte and text types inappropriately, whereas in Python 2, code that mixes those types
may work until a user causes an exception by entering non-ASCII input.
Python 3 forces programmers to proactively define a strategy for
working with strings in their program so that they don?t mix text and byte strings unintentionally.
Ansible uses different strategies for working with strings in the code on the control node, in
:ref:
modules <module_string_strategy>
, and in
module_utils
code.
Control node string strategy: the Unicode Sandwich
?
Until recently
ansible-core
supported Python 2.x and followed this strategy, known as the Unicode Sandwich (named
after Python 2?s
unicode
text type). For Unicode Sandwich we know that
at the border of our code and the outside world (for example, file and network IO,
environment variables, and some library calls) we are going to receive bytes.
We need to transform these bytes into text and use that throughout the
internal portions of our code. When we have to send those strings back out to
the outside world we first convert the text back into bytes.
To visualize this, imagine a ?sandwich? consisting of a top and bottom layer
of bytes, a layer of conversion between, and all text type in the center.
For compatibility reasons you will see a bunch of custom functions we developed (
to_text
/
to_bytes
/
to_native
)
and while Python 2 is not a concern anymore we will continue to use them as they apply for other cases that make
dealing with unicode problematic.
While we will not be using it most of it anymore, the documentation below is still useful for those developing modules
that still need to support both Python 2 and 3 simultaneously.
Unicode Sandwich common borders: places to convert bytes to text in control node code
?
This is a partial list of places where we have to convert to and from bytes
when using the Unicode Sandwich string strategy. It is not exhaustive but
it gives you an idea of where to watch for problems.
Reading and writing to files
?
In Python 2, reading from files yields bytes. In Python 3, it can yield text.
To make code that?s portable to both we don?t make use of Python 3?s ability
to yield text but instead do the conversion explicitly ourselves. For example:
from
ansible.module_utils.common.text.converters
import
to_text

```
with
open
(
'filename-with-utf8-data.txt'
,
'rb'
)
as
my_file
:
b_data
=
my_file
.
read
()
try
:
data
=
to_text
(
b_data
,
errors
=
'surrogate_or_strict'
)
except
UnicodeError
:
# Handle the exception gracefully -- usually by displaying a good
# user-centric error message that can be traced back to this piece
# of code.
pass
```

Note

Much of Ansible assumes that all encoded text is UTF-8. At some point, if there is demand for other encodings we may change that, but for now it is safe to assume that bytes are UTF-8.

Writing to files is the opposite process:

```
from
ansible.module_utils.common.text.converters
import
to_bytes
with
open
(
'filename.txt'
,
'wb'
)
as
my_file
:
```

my_file
.
write
(
to_bytes
(
some_text_string
))
Note that we don?t have to catch
UnicodeError
here because we?re
transforming to UTF-8 and all text strings in Python can be transformed back
to UTF-8.
Filesystem interaction
?
Dealing with file names often involves dropping back to bytes because on UNIX-like
systems file names are bytes. On Python 2, if we pass a text string to these
functions, the text string will be converted to a byte string inside of the
function and a traceback will occur if non-ASCII characters are present. In
Python 3, a traceback will only occur if the text string can?t be decoded in
the current locale, but it is still good to be explicit and have code which
works on both versions:
import
os.path
from
ansible.module_utils.common.text.converters
import
to_bytes
filename
=
u
'/var/tmp/????.txt'
f
=
open
(
to_bytes
(
filename
),
'wb'
)
mtime
=
os
.
path
.
getmtime
(
to_bytes
(
filename
))

```
b_filename
=
os
.
path
.
expandvars
(
to_bytes
(
filename
))
if
os
.
path
.
exists
(
to_bytes
(
filename
)):
pass
```

When you are only manipulating a filename as a string without talking to the filesystem (or a C library which talks to the filesystem) you can often get away without converting to bytes:

```
import
os.path
os
.
path
.
join
(
u
'/var/tmp/café'
,
u
'????'
)
os
.
path
.
split
(
u
'/var/tmp/café/????'
)
```

On the other hand, if the code needs to manipulate the file name and also talk to the filesystem, it can be more convenient to transform to bytes right away and manipulate in bytes.

Warning

Make sure all variables passed to a function are the same type.
If you?re working with something like
os.path.join()
which takes
multiple strings and uses them in combination, you need to make sure that
all the types are the same (either all bytes or all text). Mixing
bytes and text will cause tracebacks.
Interacting with other programs
?
Interacting with other programs goes through the operating system and
C libraries and operates on things that the UNIX kernel defines. These
interfaces are all byte-oriented so the Python interface is byte oriented as
well. On both Python 2 and Python 3, byte strings should be given to Python?s
subprocess library and byte strings should be expected back from it.
One of the main places in Ansible?s control node code that we interact with
other programs is the connection plugins?
exec_command
methods. These
methods transform any text strings they receive in the command (and arguments
to the command) to execute into bytes and return stdout and stderr as byte strings
Higher level functions (like action plugins?
_low_level_execute_command
)
transform the output into text strings.
Module string strategy: Native String
?
In modules we use a strategy known as Native Strings. This makes things
easier on the community members who maintain so many of Ansible?s
modules, by not breaking backwards compatibility by
mandating that all strings inside of modules are text and converting between
text and bytes at the borders.
Native strings refer to the type that Python uses when you specify a bare
string literal:
"This is a native string"
In Python 2, these are byte strings. In Python 3 these are text strings. Modules should be
coded to expect bytes on Python 2 and text on Python 3.
Module_utils string strategy: hybrid
?
In
module_utils
code we use a hybrid string strategy. Although Ansible?s
module_utils
code is largely like module code, some pieces of it are
used by the control node as well. So it needs to be compatible with modules
and with the control node?s assumptions, particularly the string strategy.
The module_utils code attempts to accept native strings as input
to its functions and emit native strings as their output.
In
module_utils
code:
Functions
must
accept string parameters as either text strings or byte strings.
Functions may return either the same type of string as they were given or the native string type for the Python version

they are run on.
Functions that return strings
must
document whether they return strings of the same type as they were given or native strings.
Module-utils functions are therefore often very defensive in nature.
They convert their string parameters into text (using
ansible.module_utils.common.text.converters.to_text
)
at the beginning of the function, do their work, and then convert
the return values into the native string type (using
ansible.module_utils.common.text.converters.to_native
)
or back to the string type that their parameters received.
Tips, tricks, and idioms for Python 2/Python 3 compatibility
?
Use forward-compatibility boilerplate
?
Use the following boilerplate code at the top of all python files
to make certain constructs act the same way on Python 2 and Python 3:
Make coding more python3-ish
from
__future__
import
(
absolute_import
,
division
,
print_function
)
__metaclass__
=
type
__metaclass__
=
type
makes all classes defined in the file into new-style
classes without explicitly inheriting from
object
.
The
__future__
imports do the following:
absolute_import
:
Makes imports look in
sys.path
for the modules being
imported, skipping the directory in which the module doing the importing
lives. If the code wants to use the directory in which the module doing
the importing, there?s a new dot notation to do so.
division
:
Makes division of integers always return a float. If you need to

find the quotient use
x
//
y
instead of
x
/
y
.
print_function
:
Changes
print
from a keyword into a function.
See also
PEP 0328: Absolute Imports
PEP 0238: Division
PEP 3105: Print function
Prefix byte strings with
b_
?
Since mixing text and bytes types leads to tracebacks we want to be clear
about what variables hold text and what variables hold bytes. We do this by
prefixing any variable holding bytes with
b_
. For example:
filename
=
u
'/var/tmp/café.txt'
b_filename
=
to_bytes
(
filename
)
with
open
(
b_filename
)
as
f
:
data
=
f
.
read
()
We do not prefix the text strings instead because we only operate
on byte strings at the borders, so there are fewer variables that need bytes
than text.
Import Ansible?s bundled Python

six
library
?
The third-party Python
six
library exists
to help projects create code that runs on both Python 2 and Python 3. Ansible includes a version of the library in module_utils so that other modules can use it without requiring that it is installed on the remote system. To make use of it, import it like this:
from
ansible.module_utils
import
six
Note
Ansible can also use a system copy of six
Ansible will use a system copy of six if the system copy is a later version than the one Ansible bundles.
Handle exceptions with
as
?
In order for code to function on Python 2.6+ and Python 3, use the new exception-catching syntax which uses the
as
keyword:
try
:
a
=
2
/
0
except
ValueError
as
e
:
module
.
fail_json
(
msg
=
"Tried to divide by zero:
%s
"
%
e
)
Do
not
use the following syntax as it will fail on every version of Python 3:
try:
    a = 2/0

except ValueError, e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
Update octal numbers
?
In Python 2.x, octal literals could be specified as
0755
. In Python 3,
octals must be specified as
0o755
.
String formatting for control node code
?
Use
str.format()
for Python 2.6 compatibility
?
Starting in Python 2.6, strings gained a method called
format()
to put
strings together. However, one commonly used feature of
format()
wasn?t
added until Python 2.7, so you need to remember not to use it in Ansible code:
Does not work in Python 2.6!
new_string
=
"Dear
{}
, Welcome to
{}
"
.
format
(
username
,
location
)
Use this instead
new_string
=
"Dear
{0}
, Welcome to
{1}
"
.
format
(
username
,
location
)
Both of the format strings above map positional arguments of the

format()
method into the string. However, the first version doesn?t work in
Python 2.6. Always remember to put numbers into the placeholders so the code
is compatible with Python 2.6.
See also
Python documentation on format strings:
format strings in 2.6
format strings in 3.x
Use percent format with byte strings
?
In Python 3.5 and later, byte strings do not have a
format()
method. However, it
does have support for the older, percent-formatting.
b_command_line
=
b
'ansible-playbook --become-user
%s
-K
%s
'
%
(
user
,
playbook_file
)
Previous
Next

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/ecosystem.html

Ansible Ecosystem | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible Ecosystem
The Ansible ecosystem contains a wide range of open source projects managed by Red Hat and a vibrant community of contributors.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Awesome Ansible
A collaborative curated list of awesome Ansible resources, tools, roles, tutorials and other related content.
Awesome Ansible list
Ansible collections
Ansible collections offer distributions of playbooks, roles, modules, and plugins.
Collection index
Find out how to use collections
Learn how to contribute to collections
Index of all modules and plugins
Ansible AWX
AWX provides a web-based user interface, REST API, and task engine built on top of Ansible.
Ansible AWX documentation
AWX Operator
Ansible AWX Operator offers built-in intelligence and operational best practices for deploying on Kubernetes environments.
AWX Operator documentation
Ansible Builder
Ansible Builder lets you create Execution Environments, which are container images that act as Ansible control nodes.
Ansible Builder documentation
Ansible Compat
Compat is a Python package that assists with compatibility between different Ansible releases, starting at version 2.9.
Ansible Compat documentation
Ansible Core
Ansible Core is the language and runtime that powers automation. It also provides command-line tools such as Ansible Test.
Ansible Core documentation
Ansible Test documentation

Ansible Creator
Ansible Creator is a Command-Line Interface (CLI) tool designed for effortlessly scaffolding all your Ansible content.
Ansible Creator documentation
Ansible Development Environment
A pip-like install for Ansible collections.
Ansible Development Environment documentation
Ansible Development Tools
Ansible Development Tools (ADT) streamlines the setup and usage of several tools for creating Ansible content.
Ansible Development Tools documentation
Event-Driven Ansible Server
Event-Driven Ansible Server offers scalable and flexible automation that can subscribe to a wide variety of event sources.
Event-Driven Ansible Server documentation
Edge Automation
Edge provides tooling and collections to run automation jobs on device endpoints at the very edge of your infrastructure.
Osbuild Composer Collection
Common Industrial Protocol (CIP) Collection
FDO Collection
MicroShift Collection
Galaxy NG
Galaxy NG jumpstarts automation projects with Ansible community content.
Galaxy NG documentation
Ansible Lint
Lint improves code quality through proven best practices, patterns, and behaviors so that your Ansible content results in reliable and consistent automation.
Ansible Lint documentation
Molecule
Molecule helps you develop and test Ansible roles.
Molecule documentation
Ansible Navigator
Ansible Navigator is a command-line tool for creating, reviewing, and troubleshooting Ansible content.
Ansible Navigator documentation
ansible-pylibssh
ansible-pylibssh provides Python bindings for Ansible with the libssh project.
ansible-pylibssh documentation
Ansible Pytest
Enables the use of Ansible in tests as well as the use of pytest as a collection unit test runner, and exposes molecule scenarios using a pytest fixture.
Ansible Pytest documentation
Ansible Rulebook
Ansible Rulebook is a command-line tool that listens to events so your automation can react when software or system states change.
Ansible Rulebook documentation
Ansible Runner
Ansible Runner provides a stable and consistent interface abstraction to Ansible.
Ansible Runner documentation
Ansible SDK
Ansible SDK is a toolkit that lets you harness the power and simplicity of Ansible automation directly from your applications.
Ansible SDK documentation
Ansible Sign
Ansible Sign is a utility for signing and verifying Ansible content.
Ansible Sign documentation
Tox Ansible

Tox Ansible is a utility designed to simplify the testing of Ansible content collections.
Tox Ansible documentation
Ansible VS Code Extension
The VS Code extention adds Ansible language support to Visual Studio Code and OpenVSX compatible editors.
Ansible VS Code Extension documentation
Ansible community package
The Ansible community package consists of ansible-core and a set of Ansible collections published as the Python `ansible` package, in tradition of the Ansible 2.9 and earlier "batteries included" releases.
PyPI page for Ansible community package
Documentation of package build process
Source code of package build
Source code of the Antsibull build tool
Antsibull Changelog
A changelog generator used by ansible-core and Ansible collections.
Documentation
Source code
Antsibull Docs
Tooling for building documenation for Ansible collections, ansible-core, and the Ansible community package.
Documentation
Source code
Python library for parsing Ansible markup
TypeScript library for parsing Ansible markup

Content from: https://docs.ansible.com/index.html

Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Ansible community documentation
Ansible offers open-source automation that is simple, flexible, and powerful.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
Ansible package docs home
Collection index
Modules and plugins index
Get started with Ansible
Understand the fundamentals of Ansible automation
Install the Ansible package
Run your first ad hoc command in a few easy steps
Users
Start writing Ansible playbooks
Learn about Ansible modules
Build inventory files to manage multiple hosts
Continue the Ansible user journey
Developers
Set up your development environment
Learn how Ansible works
Write custom modules or plugins
Continue the Ansible developer journey
Maintainers
Review community maintainer responsibilities
Understand Ansible contributor paths
Explore ways to grow community
Continue the Ansible community maintainer journey

Content from: https://docs.ansible.com/maintainers.html

Maintainers | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Maintainers
Ansible community maintainers are trusted contributors who oversee project lifecycle and overall health.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
News for maintainers
Community topics
Package inclusion requests
Learn about community maintainer responsibilities
Review community maintainer responsibilities
Backport merged pull requests to stable branches
Regularly release stable versions
Look after collection documentation
Expand community around a collection
Explore ways to grow community
Maintain good contributor documentation
Understand Ansible contributor paths
Get collections included in the Ansible package
Understand the inclusion process
Review other inclusion requests
Submit your collection for inclusion
Participate in cross-project governance
Discuss and vote on community topics
Join the Ansible community steering committee

Content from: https://docs.ansible.com/modules.html

Using Ansible modules and plugins ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Using Ansible modules and plugins
Introduction to modules
Boolean variables
Module maintenance and support
Rejecting modules
Working with plugins
Modules and plugins index
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index
Indexes of all modules and plugins

Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Using Ansible modules and plugins
Edit on GitHub
Using Ansible modules and plugins
?
Note
Making Open Source More Inclusive
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see
our CTO Chris Wright?s message
.
Welcome to the Ansible guide for working with modules, plugins, and collections.
Ansible modules are units of code that can control system resources or execute system commands.
Ansible provides a module library that you can execute directly on remote hosts or through playbooks.
You can also write custom modules.
Similar to modules are plugins, which are pieces of code that extend core Ansible functionality.
Ansible uses a plugin architecture to enable a rich, flexible, and expandable feature set.
Ansible ships with several plugins and lets you easily use your own plugins.
Introduction to modules
Boolean variables
Module maintenance and support
Maintenance
Issue Reporting
Support
Rejecting modules
Working with plugins
Action plugins
Become plugins
Cache plugins
Callback plugins
Cliconf plugins
Connection plugins
Docs fragments

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/platform.html

Red Hat Ansible Automation Platform | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Red Hat Ansible Automation Platform
Ansible Automation Platform centralizes automation infrastructure, adds enterprise-grade capabilities, and provides supported collections.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Red Hat Ansible Automation Platform
Red Hat Ansible Automation Platform provides everything needed to create, execute, and manage automation in a single subscription. From execution environments to certified collections to automation analytics, discover the features and benefits of Ansible Automation Platform. This now includes Ansible controller 4.5 documentation, except the CLI and Release notes listed below.
Visit the Red Hat Ansible Automation Platform documentation
Release notes
View the changes made since the last release of Ansible Automation Controller.
Visit the release notes
Automation Controller CLI
Learn how to use the command-line client for AWX and Ansible Automation Controller.
Visit the CLI guide
Automation Controller documentation archive
Find documentation for older versions of Automation Controller.
Visit the Automation Controller documentation archive

Sponsored by

Content from: https://docs.ansible.com/playbooks_vault.html

Protecting sensitive data with Ansible vault ? Ansible Community Documentation
Blog
Ansible community forum
Documentation
Ansible Community Documentation
Ansible
Select version:
latest
11
devel
Search docs:
Ansible getting started
Getting started with Ansible
Getting started with Execution Environments
Installation, Upgrade & Configuration
Installation Guide
Ansible Porting Guides
Using Ansible
Building Ansible inventories
Using Ansible command line tools
Using Ansible playbooks
Protecting sensitive data with Ansible vault
Ansible Vault
Managing vault passwords
Encrypting content with Ansible Vault
Using encrypted variables and files
Configuring defaults for using encrypted content
When are encrypted files made visible?
Format of files encrypted with Ansible Vault
Using Ansible modules and plugins
Using Ansible collections
Using Ansible on Windows, BSD, and z/OS UNIX
Ansible tips and tricks
Contributing to Ansible
Ansible Community Guide
Ansible Collections Contributor Guide
ansible-core Contributors Guide
Advanced Contributor Guide
Ansible documentation style guide
Extending Ansible
Developer Guide
Common Ansible Scenarios
Legacy Public Cloud Guides
Network Automation
Network Getting Started
Network Advanced Topics
Network Developer Guide
Ansible Galaxy
Galaxy User Guide
Galaxy Developer Guide
Reference & Appendices
Collection Index

Indexes of all modules and plugins
Playbook Keywords
Return Values
Ansible Configuration Settings
Controlling how Ansible behaves: precedence rules
YAML Syntax
Python 3 Support
Interpreter Discovery
Releases and maintenance
Testing Strategies
Sanity Tests
Frequently Asked Questions
Glossary
Ansible Reference: Module Utilities
Special Variables
Red Hat Ansible Automation Platform
Ansible Automation Hub
Logging Ansible output
Roadmaps
Ansible Roadmap
ansible-core Roadmaps
Ansible
Protecting sensitive data with Ansible vault
Edit on GitHub
Protecting sensitive data with Ansible vault
?
Note
Making Open Source More Inclusive
Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see
our CTO Chris Wright?s message
.
Welcome to the Ansible vault documentation.
Ansible vault provides a way to encrypt and manage sensitive data such as passwords.
This guide introduces you to Ansible vault and covers the following topics:
Managing vault passwords.
Encrypting content and files with Ansible vault.
Using encrypted variables and files.
Ansible Vault
Managing vault passwords
Choosing between a single password and multiple passwords
Managing multiple passwords with vault IDs
Storing and accessing vault passwords
Encrypting content with Ansible Vault
Encrypting individual variables with Ansible Vault
Encrypting files with Ansible Vault
Using encrypted variables and files
Passing a single password
Passing vault IDs
Passing multiple vault passwords
Using
--vault-id
without a vault ID

Last updated on Oct 08, 2025.

Content from: https://docs.ansible.com/users.html

Users | Ansible Documentation
Ansible documentation
Toggle navigation
Products
Blog
Community
Webinars and training
Try it now
Toggle navigation
Join the community
Users
Developers
Maintainers
Ansible core
Ansible ecosystem
Red Hat Ansible Automation Platform
Users
Automate the management of remote systems and control their desired state.
Got thoughts or feedback on this site? We want to hear from you!
Join us in the
Ansible Forum
or open a
GitHub issue
in the docsite repository.
Quicklinks
YAML syntax
Playbook variables
Playbook conditionals
Create automation
Start writing Ansible playbooks
Learn about Ansible modules
Build inventories
Build inventory files to manage multiple hosts
Use dynamic inventories
Organize automation projects
Use roles to structure the automation project
Use Ansible execution environments
Get started with execution environments
Build execution environments
Use Ansible tooling
Create and test playbooks with Ansible Navigator
Use Ansible Lint to validate playbooks
Install Molecule to develop and test Ansible roles
Use Ansible with Visual Studio Code and OpenVSX compatible editors
Find automation content
Start exploring Ansible Galaxy
Install and use roles
Install and use collections
Share automation content
Submit roles to an existing collection
Create a new collection
Upload a collection to Ansible Galaxy

Schedule and run automation jobs
Execute automation jobs on demand
Schedule automation jobs
Use execution environments with AWX jobs

Sponsored by