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
  - Installing a collection from a git repository at the command line
  - Specifying the collection location within the git repository
- Configuring the ansible-galaxy client
- Removing a collection

> If you install a collection manually as described in this paragraph, the collection will not be upgraded automatically when you upgrade the `ansible` package or `ansible-core`.

Installing collections in containers

You can install collections with their dependencies in containers known as Execution Environments. See `getting_started_ee_index` for details.

Installing collections with `ansible-galaxy`

By default, `ansible-galaxy collection install` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`). You do not need any further configuration. By default, Ansible installs the collection in `~/.ansible/collections` under the `ansible_collections` directory.

See `Configuring the ansible-galaxy client <galaxy_server_config>` if you are using any other Galaxy server, such as Red Hat Automation Hub.

To install a collection hosted in Galaxy:

```
ansible-galaxy collection install my_namespace.my_collection
```

To upgrade a collection to the latest available version from the Galaxy server you can use the `--upgrade` option:

```
ansible-galaxy collection install my_namespace.my_collection --upgrade
```

You can also directly use the tarball from your build:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./collections
```

You can build and install a collection from a local source directory. The `ansible-galaxy` utility builds the collection using the `MANIFEST.json` or `galaxy.yml` metadata in the directory.

```
ansible-galaxy collection install /path/to/collection -p ./collections
```

You can also install multiple collections in a namespace directory.

```
ns/
├── collection1/
│   ├── MANIFEST.json
│   └── plugins/
└── collection2/
    ├── galaxy.yml
    └── plugins/
```

```
ansible-galaxy collection install /path/to/ns -p ./collections
```

> The install command automatically appends the path `ansible_collections` to the one specified with the `-p` option unless the parent directory is already in a folder called `ansible_collections`.

When using the `-p` option to specify the install path, use one of the values configured in `COLLECTIONS_PATHS`, as this is where Ansible itself will expect to find collections. If you don't specify a path, `ansible-galaxy collection install` installs the collection to the first path defined in `COLLECTIONS_PATHS`, which by default is `~/.ansible/collections`

Installing collections with signature verification

If a collection has been signed by a `distribution server`, the server will provide ASCII armored, detached signatures to verify the authenticity of the `MANIFEST.json` before using it to verify the collection's contents. This option is not available on all distribution servers. See `distributing_collections` for a table listing the servers that support collection signing.

To use signature verification for signed collections:

1. `Configured a GnuPG keyring <galaxy_gpg_keyring>` for `ansible-galaxy`, or provide the path to the keyring with the `--keyring` option when you install the signed collection.
2. Import the public key from the distribution server into that keyring.

   ```
   gpg --import --no-default-keyring --keyring ~/.ansible/pubring.kbx my-public-key.asc
   ```

3. Verify the signature when you install the collection.

   ```
   ansible-galaxy collection install my_namespace.my_collection --keyring
   ~/.ansible/pubring.kbx
   ```

   The `--keyring` option is not necessary if you have `configured a GnuPG keyring <galaxy_gpg_keyring>`.
4. Optionally, verify the signature at any point after installation to prove the collection has not been tampered with. See `verify_signed_collections` for details.

You can also include signatures in addition to those provided by the distribution server. Use the `--signature` option to verify the collection's `MANIFEST.json` with these additional signatures. Supplemental signatures should be provided as URIs.

```
ansible-galaxy collection install my_namespace.my_collection --signature
https://examplehost.com/detached_signature.asc --keyring ~/.ansible/pubring.kbx
```

GnuPG verification only occurs for collections installed from a distribution server. User-provided signatures are not used to verify collections installed from Git repositories, source directories, or URLs/paths to tar.gz files.

You can also include additional signatures in the collection `requirements.yml` file under the `signatures` key.

```
# requirements.yml
collections:
  - name: ns.coll
    version: 1.0.0
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc
```

See `collection requirements file <collection_requirements_file>` for details on how to install collections with this file.

By default, verification is considered successful if a minimum of 1 signature successfully verifies the collection. The number of required signatures can be configured with `--required-valid-signature-count` or `GALAXY_REQUIRED_VALID_SIGNATURE_COUNT`. All signatures can be required by setting the option to `all`. To fail signature verification if no valid signatures are found, prepend the value with `+`, such as `+all` or `+1`.

```
export ANSIBLE_GALAXY_GPG_KEYRING=~/.ansible/pubring.kbx
export ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT=2
ansible-galaxy collection install my_namespace.my_collection --signature
https://examplehost.com/detached_signature.asc --signature
file:///path/to/local/detached_signature.asc
```

Certain GnuPG errors can be ignored with `--ignore-signature-status-code` or `GALAXY_REQUIRED_VALID_SIGNATURE_COUNT`. `GALAXY_REQUIRED_VALID_SIGNATURE_COUNT` should be a list, and `--ignore-signature-status-code` can be provided multiple times to ignore multiple additional error status codes.

This example requires any signatures provided by the distribution server to verify the collection except if they fail due to NO_PUBKEY:

```
export ANSIBLE_GALAXY_GPG_KEYRING=~/.ansible/pubring.kbx
export ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT=all
ansible-galaxy collection install my_namespace.my_collection --ignore-signature-status-code
NO_PUBKEY
```

If verification fails for the example above, only errors other than NO_PUBKEY will be displayed.

If verification is unsuccessful, the collection will not be installed. GnuPG signature verification can be disabled with `--disable-gpg-verify` or by configuring `GALAXY_DISABLE_GPG_VERIFY`.

Installing an older version of a collection

You can only have one version of a collection installed at a time. By default, `ansible-galaxy` installs the latest available version. If you want to install a specific version, you can add a version range identifier. For example, to install the 1.0.0-beta.1 version of the collection:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

You can specify multiple range identifiers separated by `,`. Use single quotes so the shell passes the entire command, including `>`, `!`, and other operators, along. For example, to install the most recent version that is greater than or equal to 1.0.0 and less than 2.0.0:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Ansible will always install the most recent version that meets the range identifiers you specify. You can use the following range identifiers:

- `*`: The most recent version. This is the default.
- `!=`: Not equal to the version specified.
- `==`: Exactly the version specified.
- `>=`: Greater than or equal to the version specified.
- `>`: Greater than the version specified.
- `<=`: Less than or equal to the version specified.
- `<`: Less than the version specified.

> By default, `ansible-galaxy` ignores pre-release versions. To install a pre-release version, you must use the `==` range identifier to require it explicitly.

Install multiple collections with a requirements file

You can set up a `requirements.yml` file to install multiple collections in one command. This file is a YAML file in the format:

```
---
collections:
# With just the collection name
- my_namespace.my_collection

# With the collection name, version, and source options
- name: my_namespace.my_other_collection
  version: ">=1.2.0" # Version range identifiers (default: ``*``)
  source: ... # The Galaxy URL to pull the collection from (default: ``--api-server`` from
cmdline)
```

You can specify the following keys for each collection entry:

> - `name`
> - `version`
> - `signatures`
> - `source`
> - `type`

The `version` key uses the same range identifier format documented in `collections_older_version`.

The `signatures` key accepts a list of signature sources that are used to supplement those found on the Galaxy server during collection installation and `ansible-galaxy collection verify`. Signature sources should be URIs that contain the detached signature. The `--keyring` CLI option must be provided if signatures are specified.

Signatures are only used to verify collections on Galaxy servers. User-provided signatures are not used to verify collections installed from git repositories, source directories, or URLs/paths to tar.gz files.

```
collections:
  - name: namespace.name
    version: 1.0.0
    type: galaxy
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc
```

The `type` key can be set to `file`, `galaxy`, `git`, `url`, `dir`, or `subdirs`. If `type` is omitted, the `name` key is used to implicitly determine the source of the collection.

When you install a collection with `type: git`, the `version` key can refer to a branch or to a git commit-ish object (commit or tag). For example:

```
collections:
  - name: https://github.com/organization/repo_name.git
    type: git
    version: devel
```

You can also add roles to a `requirements.yml` file, under the `roles` key. The values follow the same format as a requirements file used in older Ansible releases.

```
---
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: geerlingguy.php_roles
    version: ">=0.9.3"
    source: https://galaxy.ansible.com
```

To install both roles and collections at the same time with one command, run the following:

```
$ ansible-galaxy install -r requirements.yml
```

Running `ansible-galaxy collection install -r` or `ansible-galaxy role install -r` will only install collections, or roles respectively.

> Installing both roles and collections from the same requirements file will not work when specifying a custom collection or role install path. In this scenario the collections will be skipped and the command will process each like `ansible-galaxy role install` would.

Downloading a collection for offline use

To download the collection tarball from Galaxy for offline use:

1. Navigate to the collection page.
2. Click on `Download tarball`.

You may also need to manually download any dependent collections.

Installing collections adjacent to playbooks

You can install collections locally next to your playbooks inside your project instead of in a global location on your system or on AWX.

Using locally installed collections adjacent to playbooks has some benefits, such as:

- Ensuring that all users of the project use the same collection version.
- Using self-contained projects makes it easy to move across different environments. Increased portability also reduces overhead when setting up new environments. This is a benefit when deploying Ansible playbooks in cloud environments.
- Managing collections locally lets you version them along with your playbooks.
- Installing collections locally isolates them from global installations in environments that have multiple projects.

Here is an example of keeping a collection adjacent to the current playbook, under a `collections/ansible_collections/` directory structure.

```
./
├── play.yml
├── collections/
│   └── ansible_collections/
│               └── my_namespace/
│                   └── my_collection/<collection structure lives here>
```

See `collection_structure` for details on the collection directory structure.

Installing a collection from source files

Ansible can also install from a source directory in several ways:

```
collections:
  # directory containing the collection
  - name: ./my_namespace/my_collection/
    type: dir

  # directory containing a namespace, with collections as subdirectories
  - name: ./my_namespace/
    type: subdirs
```

Ansible can also install a collection collected with `ansible-galaxy collection build` or downloaded from Galaxy for offline use by specifying the output file directly:

```
collections:
  - name: /tmp/my_namespace-my_collection-1.0.0.tar.gz
    type: file
```

> Relative paths are calculated from the current working directory (where you are invoking `ansible-galaxy install -r` from). They are not taken relative to the `requirements.yml` file.

Installing a collection from a Git repository

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 git@github.com: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
```

> 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 <https://git-scm.com/docs/git-config#Documentation/git-config.txt-urlltbasegtpushInsteadOf> 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/,7b60ddc245bc416b72d8
ea6ed7b799885110f5e5
```

Configuring the `ansible-galaxy` client

By default, `ansible-galaxy` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under `galaxy_server`).

You can use either option below to configure `ansible-galaxy collection` to use other servers (such as a custom Galaxy server):

- Set the server list in the `galaxy_server_list` configuration option in `ansible_configuration_settings_locations`.
- Use the `--server` command line argument to limit to an individual server.

To configure a Galaxy server list in `ansible.cfg`:

1. Add the `server_list` option under the `[galaxy]` section to one or more server names.
2. Create a new section for each server name.
3. Set the `url` option for each server name.
4. Optionally, set the API token for each server name. Go to https://galaxy.ansible.com/ui/token and click `Show API key`.

> The `url` option for each server name must end with a forward slash `/`. If you do not set the API token in your Galaxy server list, use the `--api-key` argument to pass in the token to the `ansible-galaxy collection publish` command.

The following example shows how to configure multiple servers:

```
[galaxy]
server_list = my_org_hub, release_galaxy, test_galaxy, my_galaxy_ng

[galaxy_server.my_org_hub]
url=https://automation.my_org/
username=my_user
password=my_pass

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token

[galaxy_server.test_galaxy]
url=https://galaxy-dev.ansible.com/
token=my_test_token

[galaxy_server.my_galaxy_ng]
url=http://my_galaxy_ng:8000/api/automation-hub/
auth_url=http://my_keycloak:8080/auth/realms/myco/protocol/openid-connect/token
client_id=galaxy-ng
```

```
token=my_keycloak_access_token
```

> You can use the `--server` command line argument to select an explicit Galaxy server in the `server_list` and the value of this argument should match the name of the server. To use a server not in the server list, set the value to the URL to access that server (all servers in the server list will be ignored). Also you cannot use the `--api-key` argument for any of the predefined servers. You can only use the `api_key` argument if you did not define a server list or if you specify a URL in the `--server` argument.

Galaxy server list configuration options

The `galaxy_server_list` option is a list of server identifiers in a prioritized order. When searching for a collection, the install process will search in that order, for example, `automation_hub` first, then `my_org_hub`, `release_galaxy`, and finally `test_galaxy` until the collection is found. The actual Galaxy instance is then defined under the section `[galaxy_server.{{ id }}]` where `{{ id }}` is the server identifier defined in the list. This section can then define the following keys:

- `url`: The URL of the Galaxy instance to connect to. Required.
- `token`: An API token key to use for authentication against the Galaxy instance. Mutually exclusive with `username`.
- `username`: The username to use for basic authentication against the Galaxy instance. Mutually exclusive with `token`.
- `password`: The password to use, in conjunction with `username`, for basic authentication.
- `auth_url`: The URL of a Keycloak server 'token_endpoint' if using SSO authentication (for example, galaxyNG). Mutually exclusive with `username`. Requires `token`.
- `validate_certs`: Whether or not to verify TLS certificates for the Galaxy server. This defaults to True unless the `--ignore-certs` option is provided or `GALAXY_IGNORE_CERTS` is configured to True.
- `client_id`: The Keycloak token's client_id to use for authentication. Requires `auth_url` and `token`. The default `client_id` is cloud-services to work with Red Hat SSO.
- `timeout`: The maximum number of seconds to wait for a response from the Galaxy server.

As well as defining these server options in the `ansible.cfg` file, you can also define them as environment variables. The environment variable is in the form `ANSIBLE_GALAXY_SERVER_{{ id }}_{{ key }}` where `{{ id }}` is the upper case form of the server identifier and `{{ key }}` is the key to define. For example, you can define `token` for `release_galaxy` by setting `ANSIBLE_GALAXY_SERVER_RELEASE_GALAXY_TOKEN=secret_token`.

For operations that use only one Galaxy server (for example, the `publish`, `info`, or `install` commands). the `ansible-galaxy collection` command uses the first entry in the `server_list`, unless you pass in an explicit server with the `--server` argument.

`ansible-galaxy` can seek out dependencies on other configured Galaxy instances to support the use case where a collection can depend on a collection from another Galaxy instance.

Removing a collection

If you no longer need a collection, simply remove the installation directory from your filesystem. The path can be different depending on your operating system:

```
rm -rf ~/.ansible/collections/ansible_collections/community/general
rm -rf ./venv/lib/python3.9/site-packages/ansible_collections/community/general
```

□□□ #2
□ □□□ □□□ 2026-10-31 14:31:28 UTC
□ □□□ □□□ 2026-10-31 14:31:30 UTC