Red Hat Enterprise Linux Automation with Ansible

FEEDBACK TRANSLATIONS CERTIFICATE OF ATTENDANCE

Managing Ansible Configuration Files ☆

Objectives

After completing this section, you should be able to describe where Ansible configuration files are located, how Ansible selects them, and edit them to apply changes to default settings.

Configuring Ansible

The behavior of an Ansible installation can be customized by modifying settings in the Ansible configuration file. Ansible chooses its configuration file from one of several possible locations on the control node.

Using /etc/ansible/ansible.cfg

The ansible package provides a base configuration file located at `/etc/ansible/ansible.cfg`. This file is used if no other configuration file is found.

Using ~/.ansible.cfg

Ansible looks for a `.ansible.cfg` file in the user's home directory. This configuration is used instead of the `/etc/ansible/ansible.cfg` if it exists and if there is no `ansible.cfg` file in the current working directory.

Using ./ansible.cfg

If an `ansible.cfg` file exists in the directory in which the `ansible` command is executed, it is used instead of the global file or the user's personal file. This allows administrators to create a directory structure where different environments or projects are stored in separate directories, with each directory containing a configuration file tailored with a unique set of settings.

> **IMPORTANT**
>
> The recommended practice is to create an `ansible.cfg` file in a directory from which you run Ansible commands. This directory would also contain any files used by your Ansible project, such as an inventory and a playbook. This is the most common location used for the Ansible configuration file. It is unusual to use a `~/.ansible.cfg` or `/etc/ansible/ansible.cfg` file in practice.

Using the ANSIBLE_CONFIG environment variable

You can use different configuration files by placing them in different directories and then executing Ansible commands from the appropriate directory, but this method can be restrictive and hard to manage as the number of configuration files grows. A more flexible option is to define the location of the configuration file with the `ANSIBLE_CONFIG` environment variable. When this variable is defined, Ansible uses the configuration file that the variable specifies instead of any of the previously mentioned configuration files.

Configuration File Precedence

The search order for a configuration file is the reverse of the preceding list. The first file located in the search order is the one that Ansible selects. Ansible only uses configuration settings from the first file that it finds.

Any file specified by the `ANSIBLE_CONFIG` environment variable overrides all other configuration files. If that variable is not set, the directory in which the `ansible` command was run is then checked for an `ansible.cfg` file. If that file is not present, the user's home directory is checked for a `.ansible.cfg` file. The global `/etc/ansible/ansible.cfg` file is only used if no other configuration file is found. If the `/etc/ansible/ansible.cfg` configuration file is not present, Ansible contains defaults which it uses.

Because of the multitude of locations in which Ansible configuration files can be placed, it can be confusing which configuration file is being used by Ansible. You can run the `ansible --version` command to clearly identify which version of Ansible is installed, and which configuration file is being used.

```
[user@controlnode ~]$ ansible --version
ansible 2.9.21
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

Another way to display the active Ansible configuration file is to use the `-v` option when executing Ansible commands on the command line.

```
[user@controlnode ~]$ ansible servers --list-hosts -v
Using /etc/ansible/ansible.cfg as config file
...output omitted...
```

Ansible only uses settings from the configuration file with the highest precedence. Even if other files with lower precedence exist, their settings are ignored and not combined with those in the selected configuration file. Therefore, if you choose to create your own configuration file in favor of the global `/etc/ansible/ansible.cfg` configuration file, you need to duplicate all desired settings from that file to your own user-level configuration file. Settings not defined in the user-level configuration file remain set to the built-in defaults, even if they are set to some other value by the global configuration file.

Managing Settings in the Configuration File

The Ansible configuration file consists of several sections, with each section containing settings defined as key-value pairs. Section titles are enclosed in square brackets. For basic operation use the following two sections:

- `[defaults]` sets defaults for Ansible operation
- `[privilege_escalation]` configures how Ansible performs privilege escalation on managed hosts

For example, the following is a typical `ansible.cfg` file:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

The directives in this file are explained in the following table:

Table 2.2. Ansible Configuration

Directive	Description
`inventory`	Specifies the path to the inventory file.
`remote_user`	The name of the user to log in as on the managed hosts. If not specified, the current user's name is used.
`ask_pass`	Whether or not to prompt for an SSH password. Can be `false` if using SSH public key authentication.
`become`	Whether to automatically switch user on the managed host (typically to `root`) after connecting. This can also be specified by a play.
`become_method`	How to switch user (typically `sudo`, which is the default, but `su` is an option).
`become_user`	The user to switch to on the managed host (typically `root`, which is the default).
`become_ask_pass`	Whether to prompt for a password for your `become_method`. Defaults to `false`.

Configuring Connections

Ansible needs to know how to communicate with its managed hosts. One of the most common reasons to change the configuration file is to control which methods and users Ansible uses to administer managed hosts. Some of the information needed includes:

- The location of the inventory that lists the managed hosts and host groups
- Which connection protocol to use to communicate with the managed hosts (by default, SSH), and whether or not a nonstandard network port is needed to connect to the server
- Which remote user to use on the managed hosts; this could be `root` or it could be an unprivileged user
- If the remote user is unprivileged, Ansible needs to know if it should try to escalate privileges to `root` and how to do it (for example, by using `sudo`)
- Whether or not to prompt for an SSH password or `sudo` password to log in or gain privileges

Inventory Location

In the `[defaults]` section, the `inventory` directive can point directly to a static inventory file, or to a directory containing multiple static inventory files and dynamic inventory scripts.

```
[defaults]
inventory = ./inventory
```

Connection Settings

By default, Ansible connects to managed hosts using the SSH protocol. The most important parameters that control how Ansible connects to the managed hosts are set in the `[defaults]` section.

By default, Ansible attempts to connect to the managed host using the same user name as the local user running the Ansible commands. To specify a different remote user, set the `remote_user` parameter to that user name.

If the local user running Ansible has private SSH keys configured that allow them to authenticate as the remote user on the managed hosts, Ansible automatically logs in. If that is not the case, you can configure Ansible to prompt the local user for the password used by the remote user by setting the directive `ask_pass = true`.

```
[defaults]
inventory = ./inventory

remote_user = root
ask_pass = true
```

Assuming that you are using a Linux control node and OpenSSH on your managed hosts, if you can log in as the remote user with a password then you can probably set up SSH key-based authentication, which would allow you to set `ask_pass = false`.

The first step is to make sure that the user on the control node has an SSH key pair configured in `~/.ssh`. You can run the `ssh-keygen` command to accomplish this.

For a single existing managed host, you can install your public key on the managed host and use the `ssh-copy-id` command to populate your local `~/.ssh/known_hosts` file with its host key, as follows:

```
[user@controlnode ~]$ ssh-copy-id root@web1.example.com
The authenticity of host 'web1.example.com (192.168.122.181)' can't be established.
ECDSA key fingerprint is 70:9c:03:cd:de:ba:2f:11:98:fa:a0:b3:7c:40:86:4b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@web1.example.com's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@web1.example.com'"
and check to make sure that only the key(s) you wanted were added.
```

NOTE

You can also use an Ansible Playbook to deploy your public key to the `remote_user` account on *all* managed hosts using the `authorized_key` module.

This course has not covered Ansible Playbooks in detail yet. A play that ensures that your public key is deployed to the managed hosts' `root` accounts might read as follows:

```
- name: Public key is deployed to managed hosts for Ansible
  hosts: all

  tasks:
    - name: Ensure key is in root's ~/.ssh/authorized_hosts
      authorized_key:
        user: root
        state: present
        key: '{{ item }}'
      with_file:
        - ~/.ssh/id_rsa.pub
```

Because the managed host would not have SSH key-based authentication configured yet, you would have to run the playbook using the `ansible-playbook` command with the `--ask-pass` option in order for the command to authenticate as the remote user.

Escalating Privileges

For security and auditing reasons, Ansible might need to connect to remote hosts as an unprivileged user before escalating privileges to get administrative access as `root`. This can be set up in the `[privilege_escalation]` section of the Ansible configuration file.

To enable privilege escalation by default, set the directive `become = true` in the configuration file. Even if this is set by default, there are various ways to override it when running ad hoc commands or Ansible Playbooks. (For example, there might be times when you want to run a task or play that does not escalate privileges.)

The `become_method` directive specifies how to escalate privileges. Several options are available, but the default is to use `sudo`. Likewise, the `become_user` directive specifies which user to escalate to, but the default is `root`.

If the `become_method` mechanism chosen requires the user to enter a password to escalate privileges, you can set the `become_ask_pass = true` directive in the configuration file.

NOTE

On Red Hat Enterprise Linux 7, the default configuration of `/etc/sudoers` grants all users in the `wheel` group the ability to use `sudo` to become `root` after entering their password.

One way to enable a user (`someuser` in the following example) to use `sudo` to become `root` without a password is to install a file with the appropriate directives into the `/etc/sudoers.d` directory (owned by `root`, with octal permissions 0400):

```
## password-less sudo for Ansible user
someuser ALL=(ALL) NOPASSWD:ALL
```

Think through the security implications of whatever approach you choose for privilege escalation. Different organizations and deployments might have different trade-offs to consider.

The following example `ansible.cfg` file assumes that you can connect to the managed hosts as `someuser` using SSH key-based authentication, and that `someuser` can use `sudo` to run commands as `root` without entering a password:

```
[defaults]
inventory = ./inventory
remote_user = someuser
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Non-SSH Connections

The protocol used by Ansible to connect to managed hosts is set by default to `smart`, which determines the most efficient way to use SSH. This can be set to other values in a number of ways.

For example, there is one exception to the rule that SSH is used by default. If you do not have `localhost` in your inventory, Ansible sets up an *implicit localhost* entry to allow you to run ad hoc commands and playbooks that target `localhost`. This special inventory entry is not included in the `all` or `ungrouped` host groups. In addition, instead of using the `smart` SSH connection type, Ansible connects to it using the special `local` connection type by default.

```
[user@controlnode ~]$ ansible localhost --list-hosts
[WARNING]: provided hosts list is empty, only localhost is available

  hosts (1):
    localhost
```

The `local` connection type ignores the `remote_user` setting and runs commands directly on the local system. If privilege escalation is being used, it runs `sudo` from the user account that ran the Ansible command, not `remote_user`. This can lead to confusion if the two users have different `sudo` privileges.

If you want to make sure that you connect to `localhost` using SSH like other managed hosts, one approach is to list it in your inventory. But, this includes it in the `all` and `ungrouped` groups, which you may not want to do.

Another approach is to change the protocol used to connect to `localhost`. The best way to do this is to set the `ansible_connection` *host variable* for `localhost`. To do this, in the directory from which you run Ansible commands, create a `host_vars` subdirectory. In that subdirectory, create a file named `localhost`, containing the line `ansible_connection: smart`. This ensures that the `smart` (SSH) connection protocol is used instead of `local` for `localhost`.

You can use this the other way around as well. If you have `127.0.0.1` listed in your inventory, by default you will connect to it using `smart`. You can also create a `host_vars/127.0.0.1` file containing the line `ansible_connection: local` and it will use `local` instead.

Host variables are covered in more detail later in the course.

NOTE

You can also use *group variables* to change the connection type for an entire host group. This can be done by placing files with the same name as the group in a `group_vars` directory, and ensuring that those files contain settings for the connection variables.

For example, you might want all your Microsoft Windows managed hosts to use the `winrm` protocol and port 5986 for connections. To configure this, you could put all of those managed hosts in group `windows`, and then create a file named `group_vars/windows` containing the following lines:

```
ansible_connection: winrm
ansible_port: 5986
```

Configuration File Comments

There are two comment characters allowed by Ansible configuration files: the hash or number sign (#) and the semicolon (;).

The number sign at the start of a line comments out the entire line. It must not be on the same line with a directive.

The semicolon character comments out everything to the right of it on the line. It can be on the same line as a directive, as long as that directive is to its left.

REFERENCES

`ansible`(1), `ansible-config`(1), `ssh-keygen`(1), and `ssh-copy-id`(1) man pages

Configuration file: Ansible Documentation (https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html)

← PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH02S02) → NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH02S04)