Red Hat

Ansible Fundamentals

Managing the Inventory

Objectives

This module describes Ansible inventory concepts and how to manage a static inventory file.

Creating a Static Inventory of Managed Hosts

Red Hat

Objectives

- Implement and use Ansible inventory and host files.
- Explain the format of inventory files.
- Create an inventory file that defines a list of Linux-based managed hosts, defines groups, and assigns managed hosts to those groups.

Introduction

- An inventory defines a collection of hosts managed by Ansible.
- Hosts can be assigned to groups.
- Groups can be managed collectively.
- Groups can contain child groups.
- Hosts can be members of multiple groups.
- Variables can be set that apply to hosts and groups.

Specifying Managed Hosts with a Static Inventory

- One way to define an Ansible inventory is as a text file.
- It can be written in a number of formats -- most commonly in an INI-style or in YAML.
- This is called a *static inventory*, because the inventory needs to be manually updated.
- It is possible to create a *dynamic inventory* that is automatically generated and updated, but we will cover that later in this course.

Inventory Location

- The location of the inventory is controlled by your current Ansible configuration file
  - **ansible --version** will show you which configuration file is in use
- That file specifies the location of the inventory in its [defaults] section:

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

- If not set by the configuration, **/etc/ansible/hosts** is used

Creating an INI-Formatted Inventory File

- In its simplest form, an INI-formatted inventory file is a list of host names or IP addresses:

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

Creating an INI-Formatted Inventory File

- Host groups allow you to collectively automate a set of systems.
- In the following example, there are two groups, webservers and db_servers

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db_servers]
db1.example.com
db2.example.com
```

Creating INI-Formatted Inventory Files

- A host can be a member of multiple groups
- This allows you to organize groups in different ways depending on how you want to manage them:
    - Web servers or database servers
    - Servers in production or testing
    - Servers in the East data center or the West data center

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db_servers]
db1.example.com
db2.example.com

[east_datacenter]
web1.example.com
db1.example.com
```

```
[west_datacenter]
web2.example.com
db2.example.com

[production]
web1.example.com
web2.example.com
db1.example.com
db2.example.com

[development]
192.0.2.42
```

Special Groups and Group Names

- Two host groups always exist:
  - **all** includes every host in the inventory
  - **ungrouped** includes every host in **all** that is not a member of another group
- Group names should not include dashes, but underscores are fine
- Avoid confusion: do not give a group the same name as a host!

Defining Nested Groups

- Ansible host inventories can include groups of host groups.
- In an INI-formatted inventory, you can add nested host groups with the **:children** suffix.
- In this example, canada and usa are nested groups inside the group north_america

```
[usa]
washington01.example.com
washington02.example.com

[canada]
ontario01.example.com
ontario02.example.com

[north_america:children]
canada
usa
```

Simplifying Host Specifications with Ranges

- It is possible to specify ranges in the host names or IP addresses.
- Both numeric and alphabetic ranges can be specified.
- Ranges match all values from [START:END].
- Groups can contain child groups.
- For example:
  - **192.168.[4:7].[0:255]** matches all IPv4 addresses in the 192.168.4.0/22 network (**192.168.4.0 through 192.168.7.255**).
  - **server[01:20].example.com** matches all hosts named **server01.example.com through server20.example.com.**
  - **[a:c].dns.example.com** matches hosts named **a.dns.example.com, b.dns.example.com, and c.dns.example.com.**

Simplifying Host Specifications with Ranges

- If leading zeros are included, they are used in the pattern.

```
[usa]
washington[1:2].example.com

[canada]
ontario[01:02].example.com
```

- In this example, **ontario01.example.com** is a match but **ontario1.example.com** is not.

Alternative Inventory File Format: YAML

- Inventory files can also be expressed in YAML format.
- A comparison of an INI-formatted inventory with an identical YAML-formatted inventory:

INI

```
[usa]
washington1.example.com
washington2.example.com

[canada]
ontario01.example.com
ontario02.example.com

[north_america:children]
canada
usa
```

YAML

```
all:
  children:
    north_america:
      children:
        canada:
          hosts:
            ontario01.example.com: {}
            ontario02.example.com: {}
        usa:
          hosts:
            washington1.example.com: {}
            washington2.example.com: {}
```

Verifying the Inventory

- You can use the **ansible-inventory** command to verify the inventory
- The **-i** option can be used to check any file rather than the current inventory
- The following command will display the current inventory in YAML format:

```
ansible-inventory -y --list
```

Verifying the Inventory

- The **ansible** command can also verify a machine's presence in the inventory.

```
[user@controlnode ~]$ ansible washington1.example.com --list-hosts
  hosts (1):
    washington1.example.com
[user@controlnode ~]$ ansible washington01.example.com --list-hosts
 [WARNING]: provided hosts list is empty, only localhost is available

  hosts (0):
```

- If an inventory contains a host and a host group with the same name, the **ansible** command prints a warning and targets the host. The host group is ignored.

Managing Connection Settings and Privilege Escalation

Objectives

- Configure connections used for communicating with managed hosts.
- Use privilege escalation for playbook and play escalation.
- Explain how Ansible selects the configuration file to use and how it is applied.

Ansible’s Agentless Architecture

- Ansible does not require you to install a custom agent on managed hosts
- Protocols and software included with the operating system are leveraged
    - SSH and Python used on Linux systems
    - Other protocols used for things like Windows (Windows Remote Management and PowerShell)
- Advantages of using common, well-tested and understood tools
    - Simpler to prepare systems
    - Reduces security risks

Controlling Connections to Managed Hosts

Ansible on the control node needs some information to successfully connect to managed hosts:

- The location of the inventory file
- The connection protocol to use (by default, SSH)
- Whether a non-standard network port is needed to connect to the server
- What user it can login as
- If the user is not *root*, whether Ansible should escalate privileges to *root*
- How Ansible should become *root* (by default, with **sudo**)
- Whether to prompt for an SSH password to log in or a **sudo** password to gain privileges

You can set default selections for this information in your Ansible configuration file.

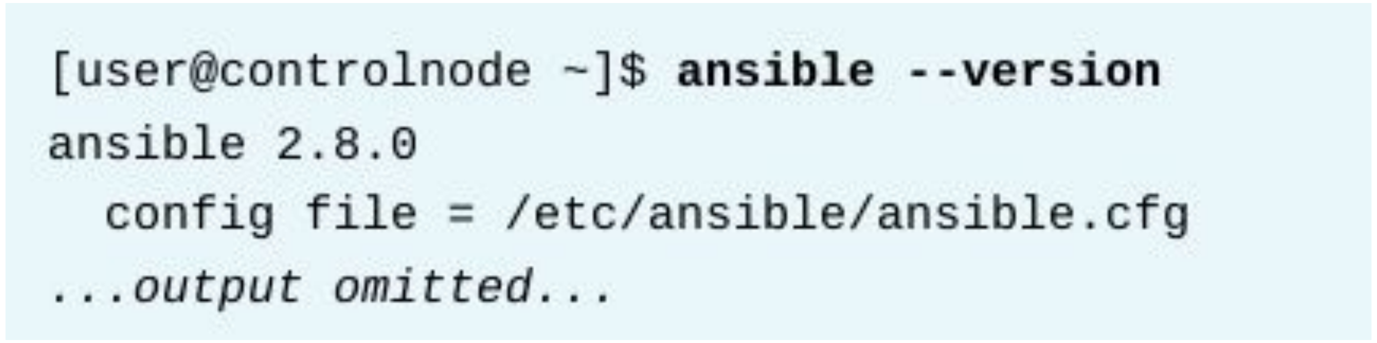
Configuring Ansible

- The behaviour of an Ansible installation can be customized by modifying settings in the Ansible configuration file. Ansible chooses its configuration from one of several possible locations:

  - The ANSIBLE_CONFIG environment variable (if set, its value is the path to the file)
  - If it is not set, Ansible will look for the configuration file in the following places:
  - **./ansible.cfg**
    - In the Ansible command's current working directory
  - **~/.ansible.cfg**
    - As a "dot file" in the user's home directory, used if there is no **./ansible.cfg** file
  - **/etc/ansible/ansible.cfg**
    - The default configuration file if no other configuration file is found

Configuration File Precedence

- **ansible --version** clearly identifies which configuration file is currently being used.



```
[user@controlnode ~]$ ansible --version
ansible 2.8.0
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

- You can use **ansible-config --version** to get the same information.

Managing Settings in the Configuration File

- **ansible.cfg** consists of several sections.
- Each section contains settings defined as key-value pairs.
- Section titles are enclosed in square brackets.
- Basic operations use two sections:
  - **[defaults]** sets defaults for Ansible operation.
  - **[privilege_escalation]** configures how Ansible performs privilege escalation on managed hosts.

Connection Settings in the Configuration File

- Settings to control your SSH connection can go in the **[defaults]** section:
    - **remote_user** specifies the user you want to use on the managed host (if you do not specify, it uses your current user name)
    - **remote_port** specifies what port sshd is using on the managed host (if you do not specify, the default is 22)
    - **ask_pass** controls whether Ansible will prompt you for the SSH password (it will not by default, assuming you are using SSH key-based authentication)

Privilege Escalation Settings in the Configuration File

- Settings to control privilege escalation can go in the **[privilege_escalation]** section:
  - **become** controls whether you will automatically use privilege escalation (default is no, and you can override this at the command line or in playbooks)
  - **become_user** controls what user on the managed host Ansible should become (default is *root*)
  - **become_method** controls how Ansible will become that user (using **sudo** by default, there are other options like **su**)
  - **become_ask_pass** controls whether to prompt you for a password for your become method (default is no)

Managing Settings in the Configuration File

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

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

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

Host-Based Connection Variables

- You can also apply settings specific to a particular host by setting connection variables
- There are several ways to do this

- One of the easiest is to place the settings in a file in the host_vars directory in the same directory as your inventory file:

  - In the example at right, there are host-specific files for server1.example.com and server2.example.com

  - The hosts should appear with those names in the inventory

- These settings override the ones in ansible.cfg

- They also have slightly different syntax and naming

```
project
├── ansible.cfg
├── host_vars
│   ├── server1.example.com
│   └── server2.example.com
└── inventory
```

Host-Based Connection and Privilege Escalation Variables

- **ansible_host** specifies a different IP address or hostname to use for the connection for this host instead of the one in the inventory
- **ansible_port** specifies the port to use for the SSH connection on this host
- **ansible_user** specifies the user you want to use on this host
- **ansible_become** specifies whether to use privilege escalation for this host
- **ansible_become_user** specifies the user to become on this host
- **ansible_become_method** specifies the privilege escalation method to use for this host

Example Host-Based Connection Variables File

- For example, the contents of **host_vars/server1.example.com** could be:

```
# connection variables for server1.example.com

ansible_host: 192.0.2.104
ansible_port: 34102
ansible_user: root
ansible_become: false
```

- These settings only affect server1.example.com in the inventory

Preparation on the Managed Host

- One of the more common choices is to set up SSH key-based authentication to an unprivileged account that can use **sudo** to become *root* without a password

- The advantage of this is that you can use a specific account that only Ansible uses, and tie that to a particular SSH private key, but still have "passwordless" authentication

- Alternatively: SSH key-based authentication to the unprivileged account, then require the **sudo** password for authentication to *root*

- Ansible allows you to select the mix of settings that works best for your security policy and stance