Business Services

orange™

Ansible Guide: Understanding Installation, Inventory Files, and Playbooks

AN INTRODUCTORY GUIDE TO ANSIBLE: UNDERSTANDING INSTALLATION, INVENTORY FILES, PLAYBOOKS, AND DYNAMIC CONFIGURATION MANAGEMENT

Part 1: Ansible Installation and Inventory File

1.1. Verifying Ansible Installation

To verify if Ansible is installed on your system, execute the following command:

```
$ ansible --version
```

1.2. Working with Inventory File

The inventory file is a configuration file where you define the host information. By default, Ansible looks for this file at /etc/ansible/hosts.

Let's open the file and add a new host:

```
$ vi /etc/ansible/hosts
```

Add the following details:

```
[webservers]
Webserver1 ansible_host=192.168.66.189 ansible_user=root
```

Here, webservers is a group, and Webserver1 is a host in that group. The ansible_host is the IP address of this host, and the ansible_user is the username to use for the SSH connection.

Part 2: Executing Ansible Ad Hoc Commands

An ad hoc command is a command which you can run individually to perform quick functions.

2.1. Ping a Managed Host

Ping a host to see if it's available by using the ping module:

```
$ ansible Webserver1 -m ping -k
```

2.2. Gather Information from a Managed Host

To gather information about a host, use the setup module:

```
$ ansible Webserver1 -m setup -k
```

2.3. Reboot a Managed Host

To reboot a host, use the reboot module:

```
$ ansible Webserver1 -m reboot -k
```

Part 3: Creating a Playbook to Install and Enable HTTPd Service

A playbook is a code file for Ansible that consists of one or multiple plays, each of which define the work to be done for a configuration on a managed host.

3.1. Creating a Playbook

1. Create a new directory to hold the playbook and inventory files:

```
$ mkdir apache-project
$ cd apache-project
$ mkdir inventories
$ cd inventories
```

2. Create a new inventory file:

```
$ vi hosts
```

Add the following details:

```
[webservers]
Webserver1 ansible_host=192.168.66.189 ansible_user=root
```

3. Create a new playbook file:

```
$ vi apache-playbook.yml
```

Copy the following into the playbook:

```
---
- hosts: webservers
  become: yes
  tasks:
  - name: install httpd
    yum:
      name: httpd
      state: installed
  - name: start and enable httpd
    systemd:
      name: httpd
      state: started
      enabled: yes
  - name: permit traffic in default zone for https service
    firewalld:
      port: 80/tcp
      permanent: yes
      state: enabled
  - name: reload service firewalld
    systemd:
      name: firewalld
      state: reloaded
```

4. Run the playbook:

```
$ ansible-playbook apache-playbook.yml -i inventories/hosts -k
```

3.2. Checking the HTTPd Service

After playbook execution, check whether the HTTPd service is installed and running on the target host:

```
$ rpm -qa httpd
$ systemctl status httpd
```

Part 4: Modifying the Playbook to Use Variables

In this part, we'll modify the existing playbook to install and start the HTTPd service using variables instead of hard-coded values. This is an important step towards making your playbook more flexible and reusable.

4.1. Update the Playbook with Variables

To start, open your playbook in a text editor:

```
$ vi apache-playbook.yml
```

Now, modify the playbook to include variables. Here's an example of how you can structure it:

```yaml
---
- hosts: webservers
  become: yes
  vars:
    package_name: httpd
    service_name: httpd
    firewall_service: https
  tasks:
  - name: install {{ package_name }}
    yum:
      name: "{{ package_name }}"
      state: installed
  - name: start and enable {{ service_name }}
    systemd:
      name: "{{ service_name }}"
      state: started
      enabled: yes
  - name: permit traffic in default zone for {{ firewall_service }} service
    firewalld:
      service: "{{ firewall_service }}"
      permanent: yes
      state: enabled
  - name: reload service firewalld
    systemd:
      name: firewalld
      state: reloaded
```

In this version of the playbook, we've introduced the vars: section where we define three variables: package_name, service_name, and firewall_service. These variables are then used in the tasks instead of having hard-coded values.

4.2. Run the Updated Playbook

You can execute the updated playbook with the same command as before:

```
$ ansible-playbook apache-playbook.yml -i inventories/hosts -k
```

4.3. Verify the Changes

Finally, verify the changes on the host:

```
$ rpm -qa httpd
$ systemctl status httpd
```

Part 5: Making Playbooks More Dynamic with Ansible Facts

Ansible facts are a way of getting data about remote systems for use in playbook variables. They're returned in a facts.d dictionary that you can reference in your playbooks.

5.1. Gather Facts from a Host

To gather facts from a host, use the setup module:

```
$ ansible Webserver1 -m setup -k
```

This command returns a large dictionary of all facts available about the host.

5.2. Using Facts in a Playbook

You can use these facts to create more dynamic playbooks. For example, you could use the ansible_distribution fact to install the correct HTTPd package for each type of Linux distribution.

Here's how you might do that:

```
---
- hosts: webservers
  become: yes
  tasks:
  - name: install httpd (CentOS)
    yum:
      name: httpd
      state: installed
    when: ansible_distribution == "CentOS"
  - name: install apache2 (Debian)
    apt:
      name: apache2
      state: installed
    when: ansible_distribution == "Debian"
```

In this playbook, the when: keyword is used to conditionally run a task. If the distribution is CentOS, it installs the httpd package. If the distribution is Debian, it installs the apache2 package.

Conclusion

This guide has covered the basics of Ansible installation, inventory files, ad hoc commands, and playbooks. You've also learned how to use variables and facts to make your playbooks more flexible and dynamic. With these tools, you can begin to automate your infrastructure and make your configurations more reliable and repeatable.