Ansible

Installation ansible on Debian

Install dependencies and configure Ansible Repository

Install ansible dependencies by running the following apt command,

```
sudo apt install -y software-properties-common
```

Once the dependencies are installed then configure PPA repository for ansible, run

```
sudo add-apt-repository --yes --update ppa:ansible/ansible
```

Now update repository by running beneath apt command.

```
sudo apt update
```

Install latest version of ansible

Now we are ready to install latest version of Ansible on Ubuntu 20.04 LTS / 21.04, run following command.

```
sudo apt install -y ansible
```

After the successful installation of Ansible, verify its version by executing the command

```
ansible --version
```

Great, above output confirms that Ansible version 2.9.6 is installed.

Directory Structure:

To create a role using the ansible-galaxy command, we can simply use the below syntax in our terminal:

```
ansible-galaxy init <ROLE_NAME>
# for example
ansible-galaxy init nginx
```

Simple Modules

Ping hosts

```
ansible <HOST_GROUP> -m ping
# for example
ansible localhost -m ping
```

Display gathered facts

```
ansible <HOST_GROUP> -m setup | less
# for example
ansible localhost -m setup | less
```

Filter gathered facts

```
ansible <HOST_GROUP> -m setup -a "filter=ansible_distribution*"
# for example
ansible localhost -m setup -a "filter=ansible_distribution*"
```

Copy SSH key manually

```
ansible <HOST_GROUP> -m authorized_key -a "user=root key='ssh-rsa
AAAA...XXX == root@hostname'"
# for example
ansible localhost -m authorized_key -a "user=root key='ssh-rsa AAAA...XXX
== root@hostname'"
```

Limit to one or more hosts

This is required when one wants to run a playbook against a host group, but only against one or more members of that group.

Limit to one host

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit "host1"
```

Limit to multiple hosts

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit "host1,host2"
```

Negated limit. NOTE: Single quotes MUST be used to prevent bash interpolation.

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit 'all:!host1'
```

Limit to host group

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit 'group1'
```

Limiting Tasks with Tags

Limit to all tags matching install

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --tags 'install'
```

Skip any tag matching sudoers

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --skip-tags 'sudoers'
```

Check for bad syntax

One can check to see if code contains any syntax errors by running the playbook.

Check for bad syntax:

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --syntax-check
```

Running a playbook in dry-run mode

Sometimes it can be useful to see what Ansible might do, but without actually changing anything.

One can run in dry-run mode like this:

```
ansible-playbook playbooks/PLAYBOOK_NAME.yml --check
```

Managing files

An ad hoc task can harness the power of Ansible and SCP to transfer many files to multiple machines in parallel. To transfer a file directly to all servers in the [atlanta] group:

```
ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

The ansible.builtin.file module allows changing ownership and permissions on files. These same options can be passed directly to the copy module as well:

```
ansible webservers -m file -a "dest=/srv/foo/a.txt mode=600"
ansible webservers -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan
group=mdehaan"
```

Install packages

ansible ad hoc command to install packages:

```
ansible localhost -a "apt install vim" --become
ansible all -i hosts.yml -a "apt install vim" --become
ansible all -i hosts.yml -m shell -a "apt install vim" --become
```

ansible fact and filter data

Facts include a large amount of variable data, filter useful data from it.

```
ansible all -i inventories/hosts.yml -m setup  -a
"filter=ansible_os_family"
ansible all -i inventories/hosts.yml -m setup  -a "filter=ansible_nodename"
ansible all -i inventories/hosts.yml -m setup  -a
"filter=ansible_interfaces"
ansible all -i inventories/hosts.yml -m setup  -a "filter=ansible_lsb"
ansible all -i inventories/hosts.yml -m setup  -a
"filter=ansible_memory_mb"
ansible all -i inventories/hosts.yml -m setup  -a
"filter=ansible_processor_vcpus"
```

Managing services

Ensure a service is started on all webservers:

```
ansible webservers -m ansible.builtin.service -a "name=httpd state=started"
```

Alternatively, restart a service on all webservers:

```
ansible webservers -m ansible.builtin.service -a "name=httpd
state=restarted"
```

Ensure a service is stopped:

```
ansible webservers -m ansible.builtin.service -a "name=httpd state=stopped"
ansible all -i inventories/hosts.yml -m service -a 'name=nginx
state=started'
ansible all -i inventories/hosts.yml -a 'systemctl status nginx'
```

Simple Ansible Hosts File Example (hosts)

```
# Define individual hosts
web1 ansible_host=192.168.1.10
```

```
db1 ansible_host=192.168.1.20

# Define a group of hosts
[webservers]
web1 ansible_user=ubuntu
web2 ansible_host=192.168.1.11 ansible_user=ubuntu

[databases]
db1 ansible_user=root
db2 ansible_host=192.168.1.21 ansible_user=root

# Define multiple groups under one larger group
[production:children]
webservers
databases

# Add specific variables for a group
[webservers:vars]
ansible_port=22
ansible_connection=ssh

[databases:vars]
ansible_port=22
ansible_connection=ssh
```

Explanation of the Example

- Individual Hosts:
  - web1 and db1 are individual hosts with their ansible_host specified (the IP address or DNS name).
  - The ansible_user specifies the user to connect as.
- Groups:
  - [webservers] and [databases] are groups of hosts. Groups allow you to apply tasks to multiple hosts at once.
  - Hosts web1, web2, db1, and db2 are part of these groups.
- Group Variables:
  - [webservers:vars] and [databases:vars] specify variables that apply to all hosts in the webservers and databases groups.
  - Variables such as ansible_port and ansible_connection define how Ansible should connect to the hosts.
- Children Groups:
  - [production:children] groups together other groups (webservers and databases) under a larger group named production.
  - This allows you to easily run playbooks against all servers in the production environment.

Complex Example with Additional Features

```
# Hosts with dynamic variables
appserver1 ansible_host=app1.mydomain.com ansible_user=deploy
```

```
ansible_port=2222
appserver2 ansible_host=app2.mydomain.com ansible_user=deploy
ansible_port=2222

# Group with multiple hosts
[appservers]
appserver1
appserver2

# Nested groups
[production:children]
appservers

# Global variables (apply to all hosts)
[all:vars]
ansible_ssh_private_key_file=~/.ssh/id_rsa
ansible_python_interpreter=/usr/bin/python3

# Host patterns (excluding appserver1)
[production:!appserver1]
```

Explanation of the Complex Example

- Dynamic Host Variables:
  - appserver1 and appserver2 are defined with dynamic host variables like ansible_host, ansible_user, and ansible_port.
- Nested Groups:
  - The [production:children] group includes the appservers group.
- Global Variables:
  - [all:vars] sets global variables that apply to all hosts in the inventory, like the SSH key and Python interpreter.
- Host Patterns:
  - [production:!appserver1] uses a pattern to target all hosts in the production group except for appserver1.

General Pattern for Ansible Ad-Hoc Commands

```
ansible <hosts> -m <module> -a "<module_options>" [-i <inventory>] [-u
<user>] [--become] [--check]
```

Breakdown of the Pattern

- : Specifies the target hosts or groups on which the command will run. This can be a single host, a group of hosts, or a pattern.
- -m : Specifies the Ansible module to be used. Modules are the building blocks for performing tasks, such as ping, shell, command, copy, yum, apt, etc.
- -a "<module_options>": Provides the options or arguments for the module. These vary depending on the module used.

- -i : (Optional) Specifies the inventory file if not using the default inventory.
- -u : (Optional) Specifies the SSH user to connect as. If omitted, Ansible will use the current user or the default SSH user.
- --become: (Optional) Specifies that the command should be run with elevated privileges (e.g., using sudo).
- --check: (Optional) Runs the command in "dry-run" mode to see what changes would be made without actually making them.

Examples of Ansible Ad-Hoc Commands

```
# Ping All Hosts
ansible all -m ping

# Check Disk Space on a Specific Group of Servers
ansible webservers -m shell -a "df -h"

# Install a Package Using apt on Debian-Based Systems
ansible all -m apt -a "name=nginx state=present" --become

# Restart a Service Using service Module
ansible webservers -m service -a "name=nginx state=restarted" --become

# Gather Facts About All Hosts
ansible all -m setup

# Execute a Command Without Using a Module
ansible all -m command -a "uptime"

# Add a User Across Multiple Servers
ansible all -m user -a "name=deploy state=present" --become

# Running Ad-Hoc Commands with Privilege Escalation
ansible all -m apt -a "name=nginx state=present" --become
```

Example of Using Ansible Facts in a Playbook

```
---
- name: Gather and use Ansible facts
  hosts: localhost
  gather_facts: yes

  tasks:
    - name: Display the operating system
      debug:
        msg: "The operating system is {{ ansible_os_family }}"

    - name: Display the IP address of the default interface
      debug:
        msg: "The default interface IP address is {{
```

```
ansible_default_ipv4.address }}"

      - name: Display the total memory in MB
        debug:
          msg: "Total memory: {{ ansible_memtotal_mb }} MB"

      - name: Display the Threads per Core
        debug:
          msg: "Threads per Core: {{ ansible_processor_threads_per_core }}"

      - name: Display Total CPU
        debug:
          msg: "Total CPU Count: {{ ansible_processor_vcpus }}"

      - name: Display all mounted filesystems
        debug:
          msg: "{{ ansible_mounts }}"
```

Explanation of the Playbook

- **gather_facts: yes:** This ensures that Ansible gathers facts about the remote system before executing tasks. This is enabled by default in most cases.
- **ansible_os_family:** This fact contains the name of the OS family (e.g., RedHat, Debian, etc.). It's used to determine which package to install (httpd for RedHat-based systems and apache2 for Debian-based systems).
- **ansible_default_ipv4.address:** This fact holds the IP address of the primary network interface.
- **ansible_hostname:** This is the hostname of the remote system, which is used to create a unique directory.
- **ansible_memtotal_mb:** This fact provides the total memory in megabytes, which is displayed using the debug module.
- **ansible_mounts:** This is a complex fact that lists all mounted filesystems, which is also displayed using the debug module.

To run this playbook, you would use the following command:

```
ansible-playbook -i hosts use_facts.yml
```

Ansible File Path Testing

Ansible Filter Testing

Ansible Gather Fact Testing

Ansible Lookup Testing

Ansible Lookup With Plugin Testing

Ansible Loop Testing

Ansible Configuration

Ansible Configuration Sample

Ansible Practice

Ansible Template Testing

Ansible Variable Precedence

Ansible Nginx