Ansible Facts and Ansible Vaults

Ansible Facts

- Facts provide certain information about a given target host.
- Facts can be cached for use in playbook executions.
- The tasks of collecting this remote system information is called gathering facts and gathered information is called facts.
- Facts are a convenient way to retrieve the state of a managed host and to determine what action to take based on that state.
- Facts use the setup module for gather the information in ad-hoc command.
- In play book by defaults its enable, but if administrator wants to disable he can able to disable by setting this parameter “gather_facts: false”.
- One way to see what facts are gathered for your managed hosts is to run a short playbook that gathers facts and uses the debug module to print the value of the ansible_facts variable.
- $ ansible -i inventory host_name -m setup.
- $ ansible active -m setup -a "filter=ansible_memory_mb"
- $ ansible active -m setup -a "filter=ansible_processor"
- $ ansible active -m setup -a "filter=ansible_os_family"
- $ ansible active -m setup -a "filter=ansible_mounts"
- $ ansible active -m setup -a "filter=ansible_architecture"

Some of the facts gathered for a managed host might include:

- The hostname
- The kernel version
- The network interfaces
- The IP addresses
- The version of the operating system
- Various environment variables
- The number of CPUs
- The available or free memory
- The available disk space

```
---
 - name: gathering facts
   hosts: active
   tasks:
    - name: Print all facts
      debug:
        var: ansible_facts                          #ansible_facts is default variable.
```

Ansible Facts and Ansible Vaults

Examples of Ansible Facts

Facts	Variable	Old form
Hostname	ansible_facts[‘hostname’]	ansible_hostname
Fully qualified domain name	ansible_facts[‘fqdn’]	ansible_fqdn
Main IPv4 address	ansible_facts[‘default_ipv4’] [‘address’]	ansible_default_ipv4[‘address’]
List of the name of all network interfaces	ansible_facts[‘interfaces’]	ansible_interfaces
Size of block devices /dev/sda	ansible_facts[‘device’] [‘sda’] [‘partitions’] [‘sda1’][‘size’]	ansible_devices[‘sda’] [‘partitions’][‘sda1’][‘size’]
List of dns server	ansible_facts [‘dns’] [‘nameserver’]	ansible_dns[‘nameserver’]
Kernel Version	ansible_facts[‘kernal’]	ansible_kernel

Note:

• ansible_facts['default_ipv4']['address'] can also be written ansible_facts.default_ipv4.address

• ansible_facts['dns']['nameservers'] can also be written ansible_facts.dns.nameservers

```
---
 - name: gathering device facts
   hosts: active
   tasks:
    - name: Print all facts
      debug:
        var: ansible_facts['devices']['nvme0n1']['partitions']['nvme0n1p1']['size']
```

```
---
 - name: Gathering info address
   hosts: active
   tasks:
    - name: Print IP address
      debug:
        msg: >
          The hostname is {{ ansible_facts.hostname }}
          and ip address is {{ ansible_facts.default_ipv4.address }}
```

Ansible Facts and Ansible Vaults

Turning Off – Ansible facts

- To disable fact gathering for a play, set the
  - gather_facts: no
  - gather_facts: false
- If you want to disable from the ansible.cfg then need to do following setting:

```
# smart - gather by default, but don't regather if already gathered

# implicit - gather by default, turn off with gather_facts: False

# explicit - do not gather by default, must say gather_facts: True

#gathering = implicit
```

Working with Custom Facts

- As an administrator you might need to gather other information apart from default information, then you need to create Custom Facts.
- These facts are integrated into the list of standard facts gathered by the setup module when it runs on the managed host.
- With the help of custom facts, Ansible can be used to adjust the behaviour of plays.
- Custom facts can be defined in a static file, formatted as an INI file or using JSON.
- Custom facts allow administrators to define certain values for managed hosts, which plays might use to populate configuration files or conditionally run tasks.
- Dynamic custom facts allow the values for these facts, or even which facts are provided, to be determined programmatically when the play is run.
- By default, the setup module loads custom facts from files and scripts in each managed host's **/etc/ansible/facts.d** directory.
- The name of each file or script must end with **.fact** in order to be used.
- Dynamic custom fact scripts must output JSON-formatted facts and must be executable.

Ansible Facts and Ansible Vaults

Create Custom Facts:

```
#!/bin/bash

pkg_name=$(httpd -version | awk 'NR==1 {print $3}')

cat << EOF
{ "HTTP_Version": "$pkg_name"
}
EOF
```

Deploy your custom facts on Ansible Node:

```
Deploy your facts on Ansible Node:
---
 - name: Installing Custom Facts
   hosts: active
   become: yes
   vars:
     remote_dir: /etc/ansible/facts.d
     fact_file: /home/ansadmin/abc.fact
   tasks:
    - name: Creating Facts Directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"

    - name: Copying Fact file
      copy:
        src: "{{ fact_file}}"
        dest: "{{ remote_dir}}"
        mode: +x
```

USING MAGIC VARIABLES

- Some variables are not facts or configured through the setup module, but are also automatically set by Ansible.
- These *magic variables* can also be useful to get information specific to a particular managed host.

Four of the most useful are:

- **hostvars:**
  - Contains the variables for managed hosts, and can be used to get the values for another managed host's variables.
  - It does not include the managed host's facts if they have not yet been gathered for that host.
  - $ ansible localhost -m debug -a "var=hostvars['localhost']"

- **group_names:**
  - Lists all groups the current managed host is in.
  - $ ansible all -m debug -a "var=group_names"

- **groups:**
  - Lists all groups and hosts in the inventory.
  - $ ansible all -m debug -a "var=groups"

- **inventory_hostname:**
  - Contains the host name for the current managed host as configured in the inventory. This may be different from the host name reported by facts for various reasons.
  - $ ansible all -m debug -a "var=inventory_hostname"

Note: There are a number of other “magic variables” as well.
For more information, see :
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedencewhere- should-i-put-a-variable.
One way to get insight into their values is to use the debug module to report on the contents of the hostvars variable for a particular host

Ansible Vaults

- Ansible may need access to sensitive data such as passwords or API keys in order to configure managed hosts.
- Normally, this information might be stored as plain text in inventory variables or other Ansible files. In that case, however, any user with access to the Ansible files or a version control system which stores the Ansible files would have access to this sensitive data. This poses an obvious security risk.
- Ansible Vault, which is included with Ansible, can be used to encrypt and decrypt any structured data file used by Ansible.
- To use Ansible Vault, a command-line tool named **ansible-vault** is used to create, edit, encrypt, decrypt, and view files. Ansible Vault can encrypt any structured data file used by Ansible.
- This might include inventory variables, included variable files in a playbook, variable files passed as arguments when executing the playbook.
- Ansible Vault does not implement its own cryptographic functions but rather uses an external Python toolkit. Files are protected with symmetric encryption using AES256 with a password as the secret key.

Ansible Facts and Ansible Vaults

Creating an Encrypted File

- To create a new encrypted file, use the **ansible-vault create *filename* command.**
- The command prompts for the new vault password and then opens a file using the default editor.
- *$ ansible-vault create secure.yml*

```
---
 - name:
   hosts: active
   become: yes
   gather_facts: true
   tasks:
   - name: test
     debug:
      msg: "{{inventory_hostname}}"
```

Viewing an Encrypted File

- You can use the ansible-vault view *filename* command to view an Ansible Vault- encrypted file without opening it for editing.
- *$ ansible-vault view secure.yml*

Editing an Existing Encrypted File

- To edit an existing encrypted file, Ansible Vault provides the ansible-vault edit *filename* command.
- This command decrypts the file to a temporary file and allows you to edit it. When saved, it copies the content and removes the temporary file.
- *$ ansible-vault edit secure.yml*

Encrypting an Existing File

- To encrypt a file that already exists, use the **ansible-vault encrypt *filename*** command.
- This command can take the names of multiple files to be encrypted as arguments.
- Use the **--output=OUTPUT_FILE** option to save the encrypted file with a new name.
- You can only use one input file with the **--output** option.
- $ ansible-vault encrypt facts.yml
- $ ansible-vault encrypt facts_ip.yml --output=output.yml

Decrypting an Existing File

- An existing encrypted file can be permanently decrypted by using the ansible-vault decrypt *filename* command. When decrypting a single file, you can use the --output option to save the decrypted file under a different name.
- *$ ansible-vault decrypt secure.yml*

Changing the Password of an Encrypted File

- You can use the ansible-vault rekey *filename* command to change the password of an encrypted file.
- This command can rekey multiple data files at once.
- It prompts for the original password and then the new password.
- *$ ansible-vault rekey secure.yml*

PLAYBOOKS AND ANSIBLE VAULT

- To run a playbook that accesses files encrypted with Ansible Vault, you need to provide the encryption password to the ansible-playbook command.
- If you do not provide the password, the playbook returns an error:
- To provide the vault password to the playbook, use the **--vault-id** option.
- *$ ansible-playbook --vault-id @prompt secure.yml*
- *$ ansible-playbook --ask-vault-pass secure.yml* (If you are using a release of Ansible earlier than version 2.4)

You can also encrypt the file where you kept your variable.

- *$ ansible-vault encrypt vars ("vars" is the file name in given example)*
- *$ $ ansible-playbook variable_file.yml --vault-id @prompt ("variable_file.yml" is the playbook name) then it will ask you the password, you need to provide file password.*