Managing Facts

Ansible Facts

Ansible facts are variables that are automatically discovered by Ansible on a managed host. Facts contain host-specific information that can be used just like regular variables in plays, conditionals, loops, or any other statement that depends on a value collected from a managed host.

Some of the facts gathered for a managed host might include

- The host name
- 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

Facts are a convenient way to retrieve the state of a managed host and to determine what action to take based on that state. For example:

- A server can be restarted by a conditional task which is run based on a fact containing the managed host's current kernel version.
- The MySQL configuration file can be customized depending on the available memory reported by a fact.
- The IPv4 address used in a configuration file can be set based on the value of a fact.

Normally, every play runs the setup module automatically before the first task in order to gather facts. This is reported as the Gathering Facts task in Ansible 2.3, or simply as setup in earlier versions of Ansible. You don't need to have a task to run setup in your play, it is automatically run for you.

In order to see what facts are gathered for managed hosts, you can run setup with an ad hoc command on those hosts. In the following example, an ad hoc command is used to run the setup module on the managed host robo2:

```
# ansible dev -m setup

robo2 | SUCCESS => {
   "ansible_facts": {
      "ansible_all_ipv4_addresses": [
         "192.168.56.5",
         "192.168.8.102"
```

The output of the ad hoc command is returned in JSON format as a hash/dictionary of variables. You can browse the output to see what facts are gathered and use them in your plays.

The following table shows some facts which might be gathered from a managed node that may be useful in a playbook:

Ansible Facts

Fact	*Variable*
Short hostname	**ansible_hostname**
Fully-qualified domain name	**ansible_fqdn**
Main IPv4 address (based on routing)	**ansible_default_ipv4.address**
A list of the names of all network interfaces	**ansible_interfaces**
Main disk first partition size (based on disk name, such as **vda**, **vdb**, and so on.)	**ansible_devices.vda.partitions.vda1.size**
A list of DNS servers	**ansible_dns.nameservers**
Version of the currently running kernel	**ansible_kernel**

Note:-

Remember that when a variable's value is a hash/dictionary, there are two syntaxes that can be used to retrieve the value. In the preceding example,

• ansible_default_ipv4.address can also be written ansible_default_ipv4['address']

• ansible_devices.vda.partitions.vda1.size can also be written ansible_devices['vda']['partitions']['vda1']['size']

• ansible_dns.nameservers can also be written ansible_dns['nameservers']

Example:-

```
---
- hosts: dev
  tasks:
  - name: Prints out the Ansible facts
    debug:
      msg: >
          The default IPv4 address of {{ ansible_fqdn }}
          is {{ ansible_default_ipv4.address }}
```

```
[ansible@robo playbook]$ ansible-playbook fact_exp.yml
SUDO password:

PLAY [dev] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [robo2.0]

TASK [Prints out the Ansible facts] ********************************************
ok: [robo2.0] => {
    "msg": "The default IPv4 address of robo2.0 is 192.168.8.102\n"
}

PLAY RECAP *********************************************************************
robo2.0                    : ok=2    changed=0    unreachable=0    failed=0
```

Fact Filters

Ansible facts contain extensive information about the system. Administrators can use Ansible filters in order to limit the results returned when gathering facts from a managed node. Filters can be used to:

- Only retrieve information about network cards.
- Only retrieve information about disks.
- Only retrieve information about users.

To use filters, the expression needs to be passed as an option, using -a 'filter=EXPRESSION'. For example, to only return information about eth0, a filter can be applied on the ansible_eth0 element:

#ansible dev -m setup -a 'filter=ansible_enp0s3'

Custom Facts

Administrators can create custom facts which are stored locally on each managed host. These facts are integrated into the list of standard facts gathered by setup when it runs on the managed host. These allow the managed host to provide arbitrary variables to Ansible which can be used to adjust the behavior of plays.

Custom facts can be defined in a static file, formatted as an INI file or using JSON. They can also be executable scripts which generate JSON output, just like a dynamic inventory script.

Custom facts allow an administrator 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 programatically when the play is run.

By default, setup 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 in .fact in order to be used. Dynamic custom fact scripts must output JSON-formatted facts and must be executable.

This is an example of a static custom facts file written in INI format. An INI-formatted custom facts file contains a top level defined by a section, followed by the key-value pairs of the facts to define:

Note

Custom fact files can not be in YAML format like a playbook. JSON format is the closest equivalent.

Custom facts are stored by **setup** in the **ansible_local** variable. Facts are organized based on the name of the file that defined them. For example, assume that the preceding custom facts are produced by a file saved as **/etc/ansible/facts.d/custom.fact** on the managed host. In that case, the value of **ansible_local['custom']['users']['user1']** is **joe**.

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, group_names, groups, inventory_hostname

There are a number of other "magic variables" as well. For more information, see http://docs.ansible.com/ansible/playbooks_variables.html

Guided Exercise: Managing Facts

ansible dev -m setup
ansible dev -m setup -a 'filter=ansible_user*'

:-Create a fact file named custom.fact. The fact file defines the package to install and the service to start on robo2.0. The file should read as follows:

cat /home/ansible/playbook/dev-vars-facts/custom.fact
[general]
package = httpd
service = httpd
state = started

:-Create the setup_facts.yml playbook to make the /etc/ansible/facts.d remote directory and to save the custom.fact file to that directory and then install the httpd package.

:- Run an ad hoc command with the setup module. Since user-defined facts are put into the ansible_local section, use a filter to display only this section. There should not be any custom facts at this point.
ansible dev -m setup -a 'filter=ansible_local'

```
robo2.0 | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {}
    },
    "changed": false
}
```

cat setup_facts.yml

```
---
- hosts: dev
  become: yes
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: Create a remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: Install the new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
        state: "{{ ansible_local.custom.general.state }}"
```

#ansible-playbook setup_facts.yml

```
[ansible@robo dev-vars-facts]$ ansible-playbook setup_facts.yml
SUDO password:

PLAY [dev] *********************************************************

TASK [Gathering Facts] *********************************************
ok: [robo2.0]

TASK [Create a remote directory] ***********************************
changed: [robo2.0]

TASK [Install the new facts] ***************************************
changed: [robo2.0]
```

#cat playbook.yml

```
---
- hosts: dev
  become: yes
  tasks:
    - name: install httpd
      yum:
        name: "{{ ansible_local.custom.general.package }}"
        state: latest
    - name: start the service
      service:
        name: "{{ ansible_local.custom.general.service }}"
        state: "{{ ansible_local.custom.general.state }}"
```

#ansible-playbook playbook.yml

```
[ansible@robo dev-vars-facts]$ ansible-playbook playbook.yml
SUDO password:

PLAY [dev] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [robo2.0]

TASK [install httpd] ***********************************************************
changed: [robo2.0]

TASK [start the service] *******************************************************
changed: [robo2.0]

PLAY RECAP *********************************************************************
robo2.0                    : ok=3    changed=2    unreachable=0    failed=0
```

#ansible dev -m command -a 'systemctl status httpd' -b

```
robo2.0 | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-04-05 08:22:37 IST; 7min ago
```

Guided Exercise: Managing Inclusions

Outcomes
You should be able to:
• Define variables and tasks in separate files.
• Include variable files and task files in playbooks.

#pwd
/home/ansible/playbook/dev-vars-inclusions

#mkdir tasks && cd tasks &&pwd

/home/ansible/playbook/dev-vars-inclusions/tasks

In the tasks directory, create the environment.yml task file. Define the two tasks that install and start the web server; use the package variable for the package name, service for the service name, and svc_state for the service state.

#cat environment.yml

```
[ansible@robo tasks]$ cat environment.yml
---
  - name: Install the {{ package }} package
    yum:
      name: "{{ package }}"
      state: latest
  - name: Start the {{ service }} service
    service:
      name: "{{ service }}"
      state: "{{ svc_state }}"
[ansible@robo tasks]$
```

cd .. && pwd
/home/ansible/playbook/dev-vars-inclusions
mkdir vars && cd vars && pwd
/home/ansible/playbook/dev-vars-inclusions/vars
In the vars directory, create the variables.yml variables file. The file defines the firewall_pkg variable in YAML format. The file should read as follows:

#cat variables.yml

```
[ansible@robo vars]$ cat variables.yml
---
firewall_pkg: firewalld
[ansible@robo vars]$
```

cd .. && pwd
/home/ansible/playbook/dev-vars-inclusions

Create and edit the main playbook, named playbook.yml. The playbook imports the tasks as well as the variables; and it installs the firewalld service and configures it.

cat playbook.yml

```
[ansible@robo dev-vars-inclusions]$ cat playbook.yml
---
- name: Configure web server
  hosts: dev
  become: yes
  vars:
    rule: http
  tasks:
    - name: Include the variables from the YAML file
      include_vars: vars/variables.yml

    - name: Include the environment file and set the variables
      include: tasks/environment.yml
      vars:
        package: httpd
        service: httpd
        svc_state: started

    - name: Install the firewall
      yum:
        name: "{{ firewall_pkg }}"
        state: latest
    - name: Start the firewall
      service:
        name: firewalld
        state: started
        enabled: true
    - name: open the port for {{ rule }}
      firewalld:
        service: "{{ rule }}"
        immediate: true
        permanent: true
        state: enabled
    - name: create index.html
      copy:
        content: "{{ ansible_fqdn }} has been customized using ansible on the {{ ansible_date_time.date }}\n"
        dest: /var/www/html/index.html
[ansible@robo dev-vars-inclusions]$
```

ansible-playbook --syntax-check playbook.yml

```
[ansible@robo dev-vars-inclusions]$ ansible-playbook --syntax-check playbook.yml

playbook: playbook.yml
```

Note: - Before executing main playbook, kindly try to uninstall {**httpd, firewalld**} if incase it got installed already.

#ansible dev -m shell -a 'yum list httpd && yum list firewalld' -b
#ansible dev -m shell -a 'yum remove httpd -y && yum remove firewalld -y' -b

ansible-playbook –syntax-check playbook.yml
ansible-playbook playbook.yml

```
[ansible@robo dev-vars-inclusions]$ ansible-playbook playbook.yml
SUDO password:

PLAY [Configure web server] ****************************************************

TASK [Gathering Facts] *********************************************************
ok: [robo2.0]

TASK [Include the variables from the YAML file] ********************************
ok: [robo2.0]

TASK [Install the httpd package] ***********************************************

changed: [robo2.0]

TASK [Start the httpd service] *************************************************
changed: [robo2.0]

TASK [Install the firewall] ****************************************************
changed: [robo2.0]

TASK [Start the firewall] ******************************************************
changed: [robo2.0]

TASK [open the port for http] **************************************************
changed: [robo2.0]

TASK [create index.html] *******************************************************
changed: [robo2.0]

PLAY RECAP *********************************************************************
robo2.0                    : ok=8    changed=6    unreachable=0    failed=0
```

Final output:-

curl 192.168.8.102

```
[ansible@robo dev-vars-inclusions]$ curl 192.168.8.102
robo2.0 has been customized using ansible on the 2019-04-12
```