Ansible Fundamentals

Using Jinja2 Templates and Filters

Deploying Files with Jinja2 Templates

Red Hat

Objectives

- Use Jinja2 templates to deploy customized files to managed hosts.

Deploying Files to Managed hosts

- There are a number of Ansible modules that can be used to deploy files, including:
  - **copy** to copy a file to the managed hosts
  - **file** to make sure a file, directory, or link exists (or does not) and has certain settings
  - **synchronize** to copy entire directories of content
- You can also edit files in place with Ansible modules
  - **lineinfile** to make sure a certain line exists in a file, for example

- However, what if a file you want to deploy needs to be customized for each managed host?
  - Use Jinja2 template files and the **template** module

Red Hat

Jinja2 Template Files

- Jinja2 template files allow you to deploy a template that contains Jinja2 variables like an Ansible Playbook
- Those variables are replaced with their values when the template is deployed
- One use case: have a complex configuration file that is customized with values set by host or group-specific inventory variables
- At right is a piece of a Jinja2 template file for /etc/ssh/sshd_config based on the value in the inventory variable **ssh_port** for the host

```
Port {{ ssh_port }}

#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

...example truncated...
```

Deploying Jinja2 Templates

- Use the **template** module to deploy a Jinja2 template file.
- It takes most of the same arguments as **copy**.
- The example task at right templates the **sshd_config.j2** file from your playbook's directory (or a **templates** directory in your playbook's directory) to **/etc/ssh/sshd_config** on the managed host, setting permissions and file ownership

```
- name: Make sure sshd_config is customized
  template:
    src: sshd_config.j2
    dest: /etc/ssh/sshd_config
    owner: root
    group: root
    mode: "0600"
    setype: etc_t
```

Jinja2 Templates and Facts

- Ansible facts are special variables that contain information unique to the managed host
- By default, they are set by an implied task at the start of the play (**Gathering Facts**)
- You can also collect facts at any time by running the **setup** module

- These facts are stored in a special variable, **ansible_facts**, structured as a dictionary
- They include network addresses, hostnames, storage configuration, operating system, and many other things
- The example play at right displays all facts for the managed host, and then just the fact that has the list of IPv4 addresses for the managed host

```
- name: Display some facts
  hosts: all

  tasks:
    - name: Display all facts
      debug:
        var: ansible_facts

    - name: Display a list of all IPv4 addresses
      debug:
        var: ansible_facts['all_ipv4_addresses']
```


Red Hat

Example: Jinja2 Templates and Facts

- At top right is a Jinja2 template, motd.j2, to be deployed as /etc/motd on the managed hosts. The fact **ansible_facts['fqdn']** will be replaced with the fully-qualified DNS name of the host.
- The /etc/motd file is templated with the task at middle right.
- The example at bottom right is what you will see on the host server1.example.com after a play containing that task runs.

```
This host is {{ ansible_facts['fqdn'] }}

Unauthorized access is prohibited.
```

```
- name: Ensure /etc/motd is correct
  template:
    src: motd.j2
    dest: /etc/motd
```

```
This host is server1.example.com

Unauthorized access is prohibited.
```

Example of a Comment

- You can use the syntax **{# *COMMENT* #}** for comments that should not appear in the final file.
- In the following example, the first line includes a comment that will not be included in the final file. The variable references in the second line are replaced with the values of the system facts being referenced.

```
{# /etc/hosts line #}
{{ ansible_facts['default_ipv4']['address'] }}   {{ ansible_facts['fqdn'] }}
```

Control Structures in Jinja2 Template Files

- The **for** statement provides a way to loop over a set of items
- **groups['all']** is a special variable that lists all the members of group *all*
- The following example Jinja2 template file uses a **for** statement to set the variable **host** to each item in the **groups['all']** list, in turn
- **hostvars['host']** is another special variable contains the facts for the current value of **host**

- The result of this template is to create a file in /etc/hosts format containing the IPv4 address and FQDN of every host in the inventory

```
{% for host in groups['all'] %}
{{ hostvars['host']['ansible_facts']['default_ipv4']['address'] }} {{ hostvars['host']['ansible_facts']['fqdn'] }}
{% endfor %}
```

Using Jinja2 Conditionals

- Jinja2 templates use the syntax **`{% EXPR %}`** for expressions or logic.
- You can use these expressions in template files but you should not use them in Ansible Playbooks.
- The **if/endif** statements allow you to put content in a deployed file based on whether another variable is set.
- In the following example, the value of the **result** variable is placed in the deployed file only if the Boolean value of the **finished** variable is **True**.

```
{# Only included if finished is True #}
{% if finished %}
{{ result }}
{% endif %}
```

Processing Variables with Jinja2 Filters

Objectives

- Use Jinja2 filters to process and reformat the values of variables.

Jinja2 Filters

- Jinja2 expressions support *filters*.
- Filters are used to modify or process the value from the variable
- Some filters are provided by the Jinja2 language itself
- Others are extensions included with Ansible as plug-ins.
- It is also possible to create custom filters, although that is beyond the scope of this course.

- Information about the filters that are available is at https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

- Filters can be incredibly useful to prepare data for use in your playbook or template.

Processing Data with Jinja2 Filters

- To apply a filter to a variable:
  - Reference the variable, but follow its name with a pipe character
  - After the pipe character, add the name of the filter you want to apply
  - Some filters might require additional arguments or options in parentheses
  - You can chain multiple filters in a pipeline

- For example, the **capitalize** filter capitalizes the first letter of a string
- Assume the value of **myname** is **james**, and you have the following Jinja2 statement:

```
{{ myname | capitalize }}
```

- **James** will be the result of the Jinja2 expansion.

Multiple filters

- The next example shows multiple filters used together
- The **unique** filter gets a unique set of items in a list, removing duplicates
- The **sort** filter sorts a list of items

- The play at right filters mylist through two filters, **unique** and **sort**

- The resulting output will be the list below:

```
- 1
- 3
- 7
- 9
```

```
- name: Multiple filter example
  hosts: localhost
  vars:
    mylist:
      - 3
      - 9
      - 1
      - 7
      - 9
  tasks:
    - name: Display sorted list of unique items
      debug:
        msg: "{{ mylist | unique | sort }}"
```

Example: The **ipaddr** Filter

- As an example, the **ipaddr** filter can perform a number of operations on IP addresses
- If passed a single IP address, it will return **True** if it is in the right format and **False** if it is not
- If passed a list of IP addresses, it will return a list of the ones that are valid.
- If you run the play at right, the results would be:

```
- 192.0.2.1
- 10.0.0.1
```

```
- name: Multiple filter example
  hosts: localhost
  vars:
    mylist:
      - 192.0.2.1
      - 10.0.0.1
      - 304.252.1.200
  tasks:
    - name: Display list of valid addresses
      debug:
        msg: "{{ mylist | ipaddr }}"
```

Example: The **ipaddr** Filter

- A more complex example actually uses ipaddr to reformat the output
- You can use an option in parentheses to tell the filter to convert a network and netmask from host and VLSN notation to CIDR network and prefix notation
- If you run the play at right, the following output will result:

```
- 192.0.2.0/24
- 10.0.0.0/25
- 10.0.0.128/25
```

- Note that this actually changed the format of the value displayed, not just its order or which items appear

```
- name: Multiple filter example
  hosts: localhost
  vars:
    mylist:
      - 192.0.2.1/255.255.255.0
      - 10.0.0.1/255.255.255.128
      - 10.0.0.200/255.255.255.128
  tasks:
    - name: Display list of CIDR networks
      debug:
        msg: "{{ mylist | ipaddr('network/prefix')
}}"
```


Red Hat

Processing Variables with Jinja2 Filters

Important

Filters do not change the value stored in the variable.
The Jinja2 expression processes that value and uses the result without changing the variable itself.

- There are a large number of filters available, both as standard filters from Jinja2 and as additional filters provided by Ansible, too many to cover in a few minutes.

- A small selection of the filters you should investigate on your own include:

  - mandatory
  - default
  - int/float
  - min/max/sum
  - first/last/length
  - unique
  - union
  - Intersect
  - difference
  - combine
  - dict2items
  - items2dict
  - lower/upper
  - capitalize
  - random
  - reverse
  - sort
  - flatten

- Learn more by reviewing the documentation at https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

Templating External Data with Lookup Plugins

Objectives

- Use lookup plugins to template external data within Jinja2 templates

Using Lookup Plugins to Import Data

- A *lookup plugin* is an Ansible extension to the Jinja2 templating language
- They import and format data from external sources for use in variables and templates
- Allows you to use the contents of a file as a value for a variable
- Allows you to look up information from other sources and put them in a template

- **ansible-doc -t lookup -l** will list all available lookup plugins

- **ansible-doc -t lookup file** will display documentation for the **file** lookup plugin

Lookup and Query

- There are two ways to call a lookup plugin:
  - **lookup** returns a string of comma-separated items
  - **query** returns an actual YAML list of items

- **query** is often easier to use without further processing:

The example at right will use the **dig** lookup plugin to look up the DNS MX records for gmail.com and returns a list where each item is one record.

It then prints the list one item at a time.

```
- name: Example of a lookup plugin in use
  hosts: all
  vars:
    mxvar: "{{ query('dig', 'gmail.com', 'qtype=MX') }}"
  tasks:
    - name: List each MX record for gmail.com
      debug:
        msg: An MX record is: {{ item }}
      loop: "{{ mxvar }}"
```

Example: File Lookups

- The **file** lookup plugin can be used to load the contents of a file into a variable
- If you provide a relative path, the plugin looks for files in the playbook's **files** directory

The play at right will look use the **authorized_key** module to copy the contents of **files/naoko.key.pub** into the **~naoko/.ssh/authorized_keys** file for user *naoko* on each managed host.

We use the lookup plugin because the value of **key** must be her actual public key, not a file name.

```
- name: Add authorized keys
  hosts: all
  vars:
    users:
      - naoko
  tasks:
    - name: Add authorized keys
      authorized_key:
        user: "{{ item }}"
        key: "{{ lookup('file', item + '.key.pub') }}"
      loop: "{{ users }}"
```

Example: Command Output Lookups with Lines as Items

- The **lines** lookup plugin will read output from a command, making each line an item in the list
- This can be useful in conjunction with filters

The example task at right uses **lines** to build a list consisting of the lines in the /etc/passwd file.

It then loops over that list, using the debug module and the **regex_replace** filter to print out the name of each user account listed in that file.

```
- name: Print the name of each account in /etc/passwd
  debug:
    msg: A user is {{ item | regex_replace(':.*$') }}
  loop: "{{ query('lines', 'cat /etc/passwd') }}"
```


Red Hat

Example: Template Lookups

- The **template** lookup plugin will take a Jinja2 template and evaluate that when setting the value.
- If you pass a relative path to the template, Ansible will look in the playbook's **templates** directory.
- For example, assume that **templates/my.template.j2** has the content:

```
Hello {{ my_name }}!
```

The example play on the right will print out the text

```
Hello class!
```

```
- name: Print "Hello class!"
  hosts: all
  vars:
    my_name: class

  tasks:
    - name: Demonstrate template lookup plugin
      debug:
        msg: "{{ lookup('template', 'my.template.j2')
}}"
```

Example: URL Lookups

- The **url** lookup plugin is useful to grab the content of a web page or the output of an API
- This example talks to an Amazon API and prints the IPv4 and IPv6 networks used by AWS

```
- name: test url lookups
  hosts: localhost
  become: no
  vars:
    amazon_ip_ranges: "{{ lookup('url', 'https://ip-ranges.amazonaws.com/ip-ranges.json', split_lines=False) }}"

  tasks:
    - name: display IPv4 ranges
      debug:
        msg: "{{ item['ip_prefix'] }}"
      loop: "{{ amazon_ip_ranges['prefixes'] }}"

    - name: display IPv6 ranges
      debug:
        msg: "{{ item['ipv6_prefix'] }}"
      loop: "{{ amazon_ip_ranges['ipv6_prefixes'] }}"
```

Learning More about Lookup Plugins

- There are many more lookup plugins available.
- Remember to use **ansible-doc -t lookup** commands to find useful documentation.
- Lookup plugins are powerful, especially once you are skilled at using variables and filters