Red Hat Enterprise Linux Automation with Ansible

FEEDBACK TRANSLATIONS CERTIFICATE OF ATTENDANCE

Deploying Custom Files with Jinja2 Templates ☆

Objectives

After completing this section, you should be able to deploy files to managed hosts that are customized by using Jinja2 templates.

Templating Files

Red Hat Ansible Automation Platform has a number of modules that can be used to modify existing files. These include `lineinfile` and `blockinfile`, among others. However, they are not always easy to use effectively and correctly.

A much more powerful way to manage files is to *template* them. With this method, you can write a template configuration file that is automatically customized for the managed host when the file is deployed, using Ansible variables and facts. This can be easier to control and is less error-prone.

Introduction to Jinja2

Ansible uses the Jinja2 templating system for template files. Ansible also uses Jinja2 syntax to reference variables in playbooks, so you already know a little bit about how to use it.

Using Delimiters

Variables and logic expressions are placed between tags, or delimiters. For example, Jinja2 templates use `{% EXPR %}` for expressions or logic (for example, loops), while `{{ EXPR }}` are used for outputting the results of an expression or a variable to the end user. The latter tag, when rendered, is replaced with a value or values, and are seen by the end user. Use `{# COMMENT #}` syntax to enclose 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['hostname'] }}
```

Building a Jinja2 template

A Jinja2 template is composed of multiple elements: data, variables, and expressions. Those variables and expressions are replaced with their values when the Jinja2 template is rendered. The variables used in the template can be specified in the `vars` section of the playbook. It is possible to use the managed hosts' facts as variables on a template.

> **NOTE**
>
> Remember that the facts associated with a managed host can be obtained using the `ansible system_hostname -i inventory_file -m setup` command.

The following example shows how to create a template for `/etc/ssh/sshd_config` with variables and facts retrieved by Ansible from managed hosts. When the associated playbook is executed, any facts are replaced by their values in the managed host being configured.

> **NOTE**
>
> A file containing a Jinja2 template does not need to have any specific file extension (for example, `.j2`). However, providing such a file extension may make it easier for you to remember that it is a template file.

```
# {{ ansible_managed }}
# DO NOT MAKE LOCAL MODIFICATIONS TO THIS FILE AS THEY WILL BE LOST

Port {{ ssh_port }}
ListenAddress {{ ansible_facts['default_ipv4']['address'] }}

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

SyslogFacility AUTHPRIV

PermitRootLogin {{ root_allowed }}
AllowGroups {{ groups_allowed }}

AuthorizedKeysFile /etc/.rht_authorized_keys .ssh/authorized_keys

PasswordAuthentication {{ passwords_allowed }}

ChallengeResponseAuthentication no

GSSAPIAuthentication yes
GSSAPICleanupCredentials no

UsePAM yes

X11Forwarding yes
UsePrivilegeSeparation sandbox

AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS

Subsystem sftp  /usr/libexec/openssh/sftp-server
```

Deploying Jinja2 Templates

Jinja2 templates are a powerful tool to customize configuration files to be deployed on the managed hosts. When the Jinja2 template for a configuration file has been created, it can be deployed to the managed hosts using the `template` module, which supports the transfer of a local file on the control node to the managed hosts.

To use the `template` module, use the following syntax. The value associated with the `src` key specifies the source Jinja2 template, and the value associated with the `dest` key specifies the file to be created on the destination hosts.

```
tasks:
  - name: template render
    template:
      src: /tmp/j2-template.j2
      dest: /tmp/dest-config-file.txt
```

> **NOTE**
>
> The `template` module also allows you to specify the owner (the user that owns the file), group, permissions, and SELinux context of the deployed file, just like the `file` module. It can also take a `validate` option to run an arbitrary command (such as `visudo -c`) to check the syntax of a file for correctness before copying it into place.
>
> For more details, see `ansible-doc template`.

Managing Templated Files

To avoid having system administrators modify files deployed by Ansible, it is a good practice to include a comment at the top of the template to indicate that the file should not be manually edited.

One way to do this is to use the "Ansible managed" string set in the `ansible_managed` directive. This is not a normal variable but can be used as one in a template. The `ansible_managed` directive is set in the `ansible.cfg` file:

```
ansible_managed = Ansible managed
```

To include the `ansible_managed` string inside a Jinja2 template, use the following syntax:

```
{{ ansible_managed }}
```

Control Structures

You can use Jinja2 control structures in template files to reduce repetitive typing, to enter entries for each host in a play dynamically, or conditionally insert text into a file.

Using Loops

Jinja2 uses the `for` statement to provide looping functionality. In the following example, the `user` variable is replaced with all the values included in the `users` variable, one value per line.

```
{% for user in users %}
      {{ user }}
{% endfor %}
```

The following example template uses a `for` statement to run through all the values in the `users` variable, replacing `myuser` with each value, except when the value is `root`.

```
{# for statement #}
{% for myuser in users if not myuser == "root" %}
User number {{ loop.index }} - {{ myuser }}
{% endfor %}
```

The `loop.index` variable expands to the index number that the loop is currently on. It has a value of 1 the first time the loop executes, and it increments by 1 through each iteration.

As another example, this template also uses a `for` statement, and assumes a `myhosts` variable has been defined in the inventory file being used. This variable would contain a list of hosts to be managed. With the following `for` statement, all hosts in the `myhosts` group from the inventory would be listed in the file.

```
{% for myhost in groups['myhosts'] %}
{{ myhost }}
{% endfor %}
```

For a more practical example, you can use this to generate an `/etc/hosts` file from host facts dynamically. Assume that you have the following playbook:

```
- name: /etc/hosts is up to date
  hosts: all
  gather_facts: yes
  tasks:
    - name: Deploy /etc/hosts
      template:
        src: templates/hosts.j2
        dest: /etc/hosts
```

The following three-line `templates/hosts.j2` template constructs the file from all hosts in the group `all`. (The middle line is extremely long in the template due to the length of the variable names.) It iterates over each host in the group to get three facts for the `/etc/hosts` file.

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

Using Conditionals

Jinja2 uses the `if` statement to provide conditional control. This allows you to put a line in a deployed file if certain conditions are met.

In the following example, the value of the `result` variable is placed in the deployed file only if the value of the `finished` variable is `True`.

```
{% if finished %}
{{ result }}
{% endif %}
```

> **IMPORTANT**
>
> You can use Jinja2 loops and conditionals in Ansible templates, but not in Ansible Playbooks.

Variable Filters

Jinja2 provides filters which change the output format for template expressions (for example, to JSON). There are filters available for languages such as YAML and JSON. The `to_json` filter formats the expression output using JSON, and the `to_yaml` filter formats the expression output using YAML.

```
{{ output | to_json }}
{{ output | to_yaml }}
```

Additional filters are available, such as the `to_nice_json` and `to_nice_yaml` filters, which format the expression output in either JSON or YAML human readable format.

```
{{ output | to_nice_json }}
{{ output | to_nice_yaml }}
```

Both the `from_json` and `from_yaml` filters expect strings in either JSON or YAML format, respectively, to parse them.

```
{{ output | from_json }}
{{ output | from_yaml }}
```

Variable Tests

The expressions used with `when` clauses in Ansible Playbooks are Jinja2 expressions. Built-in Ansible tests used to test return values include `failed`, `changed`, `succeeded`, and `skipped`. The following task shows how tests can be used inside of conditional expressions.

```
tasks:
...output omitted...
  - debug: msg="the execution was aborted"
    when: returnvalue is failed
```

REFERENCES

template - Templates a file out to a remote server — Ansible Documentation (https://docs.ansible.com/ansible/2.9/modules/template_module.html)

Variables — Ansible Documentation (https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html)

Filters — Ansible Documentation (https://docs.ansible.com/ansible/2.9/user_guide/playbooks_filters.html)

← PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH05S02) → NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH05S04)