Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

Powered by simplilearn

DevOps and Cloud Computing

Caltech | Center for Technology & Management Education

Configuration Management with Ansible and Terraform

Powerd by simplilearn

Ansible Jinja2 Template

Learning Objectives

By the end of this lesson, you will be able to:

- Describe the need of a Jinja2 template
- Define Jinja2 template
- Describe various data structures used in Jinja2 template
- Write a Jinja2 template filter

A Day in the Life of a DevOps Engineer

You are working in an organization as a DevOps consultant who manages thousands of servers.
The configurations vary from cluster to cluster. The creation of static configuration files for each of these clusters is tedious. This may not be a viable option since it will consume more time and energy.
Hence, the organization is looking for a designer-friendly and widely used templating language. The solution should have a way of evaluating template expressions and returning True or False.

To achieve all of the above with some additional features, you will be learning a few concepts in this lesson that will help find a solution for the given scenario.

Introduction to Jinja2 Template

Jinja2 Template

Jinja2 is a designer-friendly and widely used templating language.

A few examples of applications using Jinja2 are Ansible, Django, Flask, Salt, and Trac.

Jinja2 Template in Ansible

Why do we need Jinja2 templating in Ansible?

Jinja2 Template is the solution for the following tedious task:

When a company needs to manage thousands of servers, the configurations vary from cluster to cluster. The creation of static configuration files for each of these clusters is tedious. This may not be a viable option since it will consume more time and energy.

Jinja2 Template in Ansible

Here's the working of the Jinja2 template with Ansible:

Jinja2 template files store variables that can change from time to time.

1

When Playbooks are executed, these variables are replaced by actual values defined in the Ansible Playbooks.

2

This way, templating offers an efficient and flexible solution to create or alter configuration files with ease.

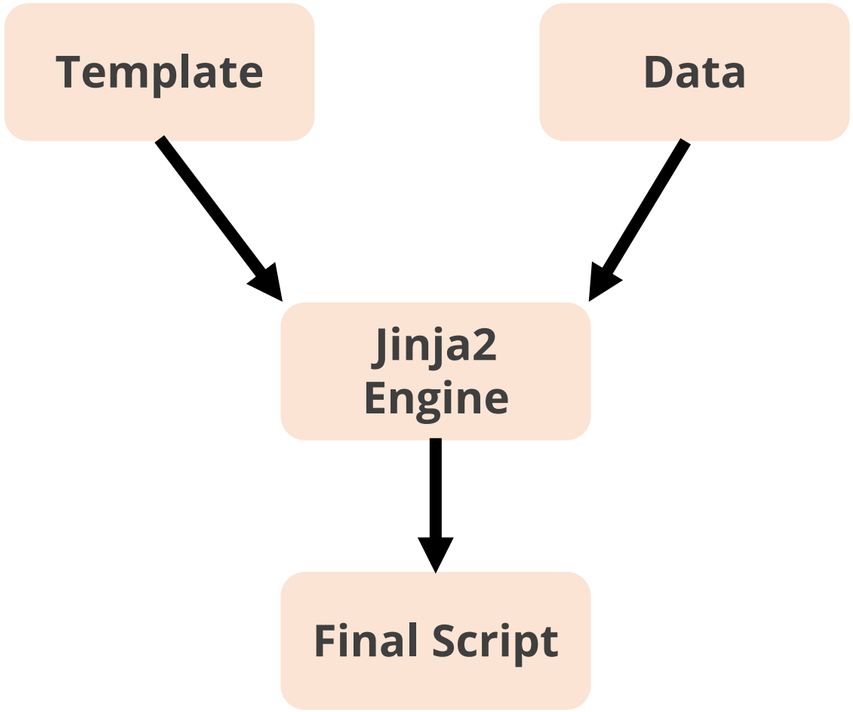
3

Where Is Templating Done?

The Jinja2 template works on the controller machine before the task is sent and executed on the target machine.

Jinja2 Template: Working

Jinja2 just requires two source ingredients: a template and data that will be used to render the final document.



Template Architecture

Jinja2 template file contains the configuration parameters.

The template file is saved with a **.j2** extension.

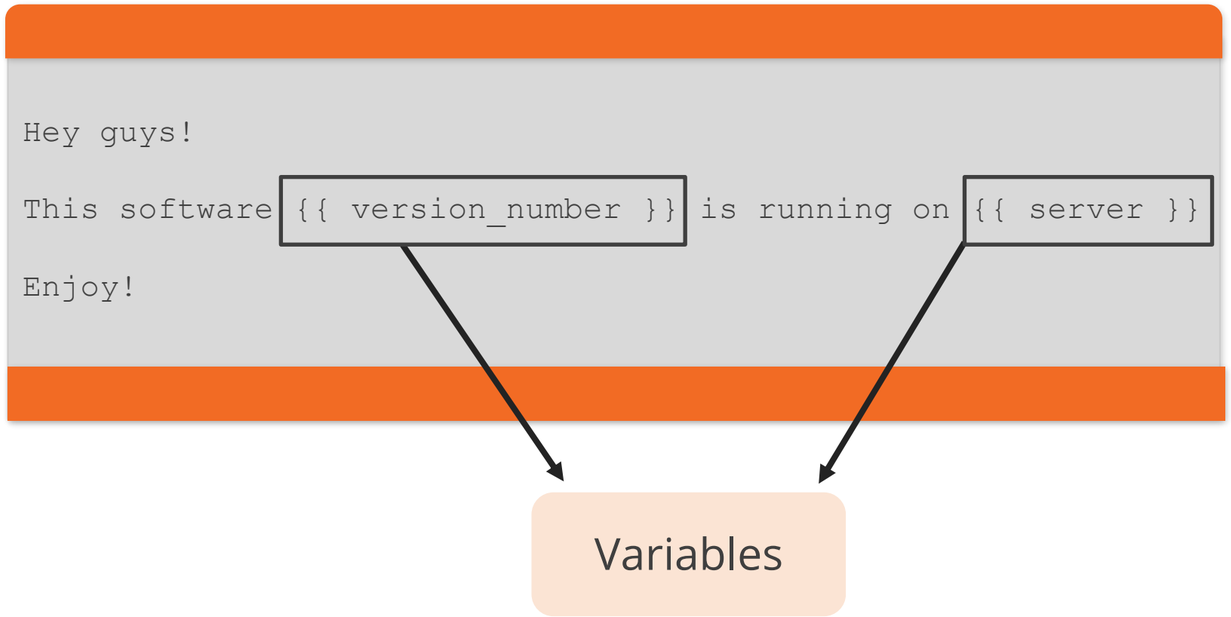
Template Architecture

Tags of Jinja2 Template

{{ }}	It is used for embedding variables which print their actual value during code execution.
{% %}	It is used for control statements such as loops and if-else.
{# #}	It is used to specify comments.
# .. ##	It is used to specify line statements.

Variables in Jinja2 Template

Jinja2 template automatically assigns the data type when a variable with a value is initialized.



These variables are defined in a playbook and will be replaced by actual values in the playbook (YAML file).

Template Filters

Filters are used to alter the appearance of output or format data by piping the variable name.

```
{{ variable | argument }}
```

Filters modify the variables.

Template Filters

Features of chained template:

- Multiple filters can be chained.
- The output of one filter is applied to the next.

```
{{ name|title|subtitle }}
```

Template Filters

The **default** filter is a way to provide a default value for an undefined variable, which will prevent Ansible from generating an error.

In-built filters can be used to iterate over the individual values of a list and perform operations.

Filters are executed on the controller and they transform the data locally.

Template Filters

Assigning default values

The “default” filter of Jinja2 can be used to provide default values for variables directly in the templates.

```
{{ missing_variable | default(2) }}
```

Similarly, variables can be made optional and mandatory using the keywords **omit** and **mandatory,** respectively.

Template Filters

Following are some of the most used Ansible built-in filters:

abs()	float()	lower()	round()	tojson()
attr()	forceescape()	map()	safe()	trim()
batch()	format()	max()	select()	truncate()
capitalize()	groupby()	min()	selectattr()	unique()
center()	indent()	pprint()	slice()	upper()
default()	int()	random()	sort()	urlencode()
dictsort()	join()	reject()	string()	urlize()
escape()	last()	rejectattr()	striptags()	wordcount()
filesizeformat()	length()	replace()	sum()	wordwrap()
first()	list()	reverse()	title()	xmlattr()

Tests

In Jinja2, Tests are a way of evaluating template expressions and returning True or False.

Tests	Filters
Jinja Tests are used for comparisons.	Filters are used for data manipulation, and have different applications in jinja.

Tests

Some features of Tests are:

01

Tests can be used in list processing filters, like map() and select() to choose items in the list.

02

Tests are always executed on the Ansible controller, and not on the target of a task, as they test local data.

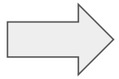
In addition to the Jinja2 Tests, Ansible offers a few more tests and even users can easily create their tests.

Test Syntax

To test a variable or expression, add ***is*** followed by the name of the test after the variable.

- Built-in Ansible tests return values such as **failed**, **changed**, **succeeded**, and **skipped**.

```
variable is test_var
```



Output

```
result is failed
```

Tests can also accept arguments.

Tests

These are some of the Ansible built-in tests.

boolean()	even()	in()	mapping()	sequence()
callable()	false()	integer()	ne()	string()
defined()	filter()	iterable()	none()	test()
divisibleby()	float()	le()	number()	true()
eq()	ge()	lower()	odd()	undefined()
escaped()	gt()	lt()	sameas()	upper()

Tests

The following is an example of matching strings against a substring or a regular expression, using the **match**, **search,** and **regex** tests:

```
vars:
  url: "http://example.com/users/foo/resources/bar"

tasks:
    - debug:
        msg: "matched pattern 1"
      when: url is
match("http://example.com/users/.*/resources/")

    - debug:
        msg: "matched pattern 2"
      when: url is search("/users/.*/resources/.*")

    - debug:
        msg: "matched pattern 3"
      when: url is search("/users/")

    - debug:
        msg: "matched pattern 4"
       when: url is regex("example.com/\w+/foo")
```

Tests

The following is an example of comparing versions:

```
vars:
    my_version: 1.2.3
tasks:
    - debug:
        msg: "my_version is higher than 1.0.0"
      when: my_version is version('1.0.0', '>')
```

If ***my_version*** is greater than or equal to ***1.0.0***, this test returns True, otherwise False.

Note

When using ***version*** in a playbook or role, don't use {{ }} as described in the FAQ.

Tests

Some examples of Tests are:

Set theory tests	To see if a list includes or is included by another list, you can use ‘subset’ and ‘superset’.
Testing if a list contains a value	The ***contains*** test is designed to work with the ***select, reject, selectattr,*** and ***rejectattr*** filters.
Testing if a list value is True	To check if any or all elements in a list are true or not, **any** and **all test** is used.
Testing size formats	The **human_readable** and **human_to_bytes** functions help in testing the right size format used in tasks.

Facts, Variable Files, and Control Structure

Ansible Facts

Ansible facts are the host-specific system data and properties to which users connect.

- Ansible facts help the admin to manage the hosts based on their current condition rather than taking the actions directly without having any information about the system's health.

A fact can be the IP address, BIOS information, a system's software information, and even hardware information.

Ansible Facts

Ansible facts use the ***setup*** module for gathering facts every time before running the playbooks.

- Ansible facts are data about the system which a user wants to configure.
- These facts make the user's Ansible system intelligent by providing the conditions for when to process some tasks.
- The user can also process by not specifying or using the Ansible facts, but that would make the job as a system admin more hectic as the script may fail or change some of the files that were never intended to be modified.

Variable File

The –vars-file option provides the path to the file containing variables.

The file path must be one of the following:

01 Absolute file path

02 Relative to the project path

03 Relative to the Ansible folder

Note

It is not possible to set variables inside a block and have them show up outside of it. This also applies to loops.

Variable File Example

This is a simple example of a variable file consisting of two dictionary variables

var.yml

```
---
  people:
    - name: Mike
      fav_colour: Blue
    - name: Kyle
      fav_colour: Yellow
  colours:
    - name: Blue
      things:
        - Sky
        - Sea
    - name: Yellow
      things:
        - Egg yolk
        - Taxi
```

Loops

This is a script of a simple playbook which uses the variable file and calls the template.

varloop.yml

```
---
  - name: Demonstrating variables in Jinja2
Loops
    hosts: localhost
    connection: local
    vars_files:
      - vars.yml
    gather_facts: no
    tasks:
      - name: Create the Jinja2 based
template
        template: src=./varloop.j2
dest=./output.txt
```

Loops

This is the Jinja2 template file which is referred to in the playbook.

varloop.j2

```
---
{% for colour in colours %}
  Colour number {{ loop.index }} is {{ colour.name }}.
{% set colour_count = 0 %}
{% for person in people if person.fav_colour == colour.name %}
{% set colour_count = colour_count + 1 %}
{% endfor %}
  Currently {{ colour_count }} people call {{ colour.name }}
their favourite.
  And the following are examples of things that are {{
colour.name }}:
{% for item in colour.things %}
    - {{ item }}
{% endfor %}
{% endfor %}
```

Loops

Output

```
---
Colour number 1 is Blue.
  Currently 0 people call Blue their favourite.
  And the following are examples of things that are Blue:
    - Sky
    - Sea

  Colour number 2 is Yellow.
  Currently 0 people call Yellow their favourite.
  And the following are examples of things that are Yellow:
    - Egg yolk
    - Taxi
```

Note

It is not possible to set variables inside a block and have them show up outside of it. This also applies to loops.

Control Structure in Jinja2

Jinja2 templating offers control statements such as loops to iterate over a list, reduce repetitive typing, enter entries for each host in a play dynamically, or conditionally insert text into a file.

01 Using Loops

02 Using Conditional control

Jinja2's syntax encloses the control structures inside the ***{% %}*** blocks.

Control Structure in Jinja2

01

Using Loops

Jinja2 uses the **for statement** to provide looping functionality.

playbook

```
---
- hosts: worker1
  become: yes
  vars:
    usernames: ['Alice', 'Bob', 'John',
'Martin']

  tasks:
    - name: Loops usage within Jinja2
templates
      template:
        src: usernames.j2
        dest:
/home/ansible_user/usernames.txt
```

usernames.j2

```
The list of users who are going to
be part of the ongoing migration
process.

{% for item in usernames %}

   {{ item }}

{% endfor %}
```

Control Structure in Jinja2

02 Using Conditional control

Jinja2 uses the ***if*** statements for conditional control.

```
{% if condition %}
    print_some_thing

{% elif condition2 %}
    execute_another_thing

{% else %}
    do_something_else

    {% endif %}
```

Conditionals

The simplest conditional statement applies to a single task.

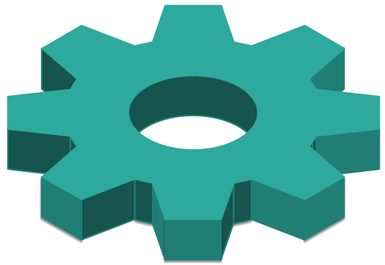
The **when** clause is a raw Jinja2 expression without double curly braces.

Ansible evaluates the **test** for all hosts when the task or playbook is executed.

Ansible runs that task on any host where the test passes.

Conditionals Based on Facts

With Conditionals based on **ansible_facts**:



Users can install a certain package only when the operating system is a particular version.

Users can skip configuring a firewall on hosts with internal IP addresses.

Users can perform cleanup tasks only when a file system is getting full.

Key Takeaways

- Jinja2 template helps in creating configuration files for servers that require different configurations.
- Filters are used to alter the appearance of the output or format data by piping the variable name.
- Jinja2 tests are a way of evaluating template expressions and returning True or False.
- It is not possible to set variables inside a block and have them show up outside of it.

Templating with Jinja2

Duration: 25 Min.

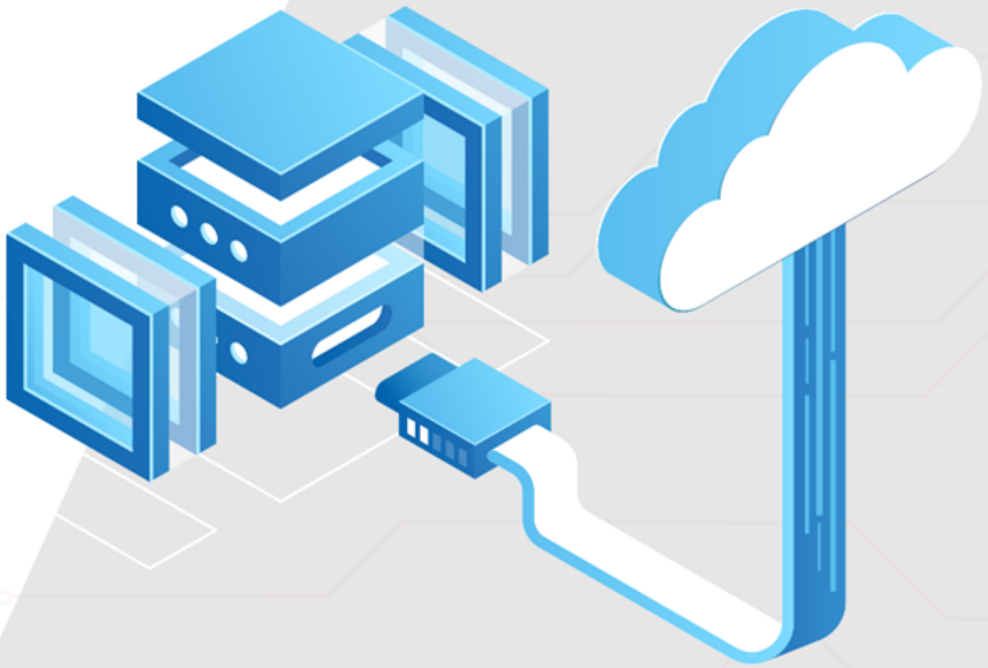
Project agenda: To use Jinja2 templates in ansible

Description: Jinja2 is a very popular and powerful Python-based template engine. It creates HTML, XML, or other markup formats that are returned to the user via an HTTP request.

Perform the following:

- Creating a new directory
- Defining a variable inside a playbook using vars
- Performing data manipulation using filters
- Using default filters
- Using filters dealing with pathnames, date and time

LESSON-END PROJECT



Thank you