**A Basic Example of Ansible Template Module**

**.j2 is extension of jinja file**

At the bare minimum, you need to have two parameters when using the Ansible module.

src: the source of the template file. This can be relative or absolute path.

dest: the destination path on the remote server

In the following task. I am using the template module on the hello\_world.j2 file. Double braces cover the variables.

- hosts: all

vars:

variable\_to\_be\_replaced: 'Hello world'

inline\_variable: 'hello again'

tasks:

- name: Ansible Template Example

template:

src: hello\_world.j2

dest: /Users/mdtutorials2/Documents/Ansible/hello\_world.txt

test.j2

hi this is my test j2 file

we want to create users and {{ var\_name }} group

hello\_world.j2

--------------

{{ variable\_to\_be\_replaced }}

This line won't be changed

Variable given as inline - {{ inline\_variable }} - :)

output - hello\_world.txt

------

Hello world

This line won't be changed

Variable given as inline - hello again - :)

mdtutorials2$ ls -lrt hello\_world.txt

-rw-r--r--  1 root  wheel  81 Oct 16 07:23 hello\_world.txt

As you can see, both variables in the hello\_world are replaced by their values.

Note: If no file name is given in the destination parameter, and only the directory path, then the name of the file will be template file name. In the above example, it would have been hello\_world.j2.

## Using ‘for’ loop structure inside Ansible template

One of the main program expression we usually use is the ‘for’ loop. It can be used to iteratively go through the values of a list, dictionary etc.

It is possible to use this in ansible templates also using the jinja2 format.

In the following example, I am looping through the value 0 to 2 using the python range function. On each iteration, a line with the variable is printed.

Jinja\_loop.j2

-------------

Ansible template for loop example

{% for i in range(3) %}

This is the {{ i }}th variable

{% endfor %}

output

------

mdtutorials2$$ cat hello\_world.txt

Ansible template for loop example

This is the 0th variable

This is the 1th variable

This is the 2th variable

But, in the above example, each iteration is printed on new lines. It is because it is retaining the whitespaces. But in some scenarios, we may want to remove the white spaces. So, how can we do that?

For such scenarios, we can use the minus sign(-) to manually strip the whitespaces including newlines. For example, in the following task, I am adding the ‘-‘ sign to the end of the ‘for’ expression. This will remove the white spaces at the end of the block.  
The resulting output shows all the variables on the same line.

Removing the whitespaces in ansible template

{% for i in range(3) %}

variable {{ i }}

{%- endfor %}

output

------

mdtutorials2 $ cat hello\_world.txt

Removing the whitespaces in ansible template

variable 0 variable 1 variable 2

### Using list variables in Ansible templates

In the below task, I am looping over the list1 variable in the template, using the for loop structure. Note that, after each iteration, a new line is also added. So the three list items will be in three lines.

- hosts: loc

vars:

list1: ['Template iterate','Template loop','Template item']

tasks:

- name: Ansible template loop example.

- template:

src: templates\_example2.j2

dest: /home/mdtutorials2/output.txt

mode: 0777

templates\_example2.j2

This is an example of template module loop with a list.

{% for item in list1 %}

{{ item }}

{% endfor %}

output.txt

This is an example of using with\_items in template module loop with a list.

Template iterate

Template loop

Template item

### Ansible template with\_items for multiple files

You can use the with\_items parameter on a dictionary to render multiple files. We are using the dictionary since the source and destination will be different for each template.

In the following example, I am rendering three templates, each with different source and destination.

- hosts: loc

tasks:

- name: Ansible template with\_items example.

template:

src: "{{ item.src }}"

dest: "{{ item.dest }}"

mode: 0777

with\_items:

- {src: 'ex.j2',dest: '/home/dnpjose/ex\_rew1.txt'}

- {src: 'ex2.j2',dest: '/home/dnpjose/ex\_rew2.txt'}

- {src: 'ex3.j2',dest: '/home/dnpjose/ex\_rew3.txt'}

## How it works

The Jinja variables and expressions indicated using the default delimiters as follows:

* {% … %} for control statements (conditions)
* {{ … }} for expressions (variables)
* {# … #} for comments (describe the task)

Here’s an example Jinja expressions:

- hosts: 127.0.0.1

vars\_files:

- vars.yml

tasks:

- name: Checking the IP address

debug:

msg: "IP address {{ ip }}"

- name: Checking OS name

debug:

msg: "OS NAME {{ os\_name }}"

Variable definitions are needed for Jinja to resolve expressions. In the above example, definitions are required for ip and os\_name.

In Ansible, more then 21 places we can declare or define variable or value to the variable, below we have shared three important places:

* Role defaults d
* Passing a YAML or JSON file with the –vars-file option
* Environment variables

## Variable files (vars\_file or vars)

Pass the path to a file containing variable definitions using the –vars-file option. The file path must be one of the following:

* Absolute file path
* Relative to the project path
* Relative to the ansible folder

When –vars-file is passed, Ansible Container checks if the path is an absolute path to a file. If not, it checks for the file relative to the project path, which is the current working directory. If the file is still not found, it looks for the file relative to the ansible folder within the project path.

Variable files can also be specified using the vars\_files directive under settings in **container.yml.** For example:

- hosts: 127.0.0.1

vars\_files:

- vars.yml

tasks:

...

This templating will helpful for many automation. It can be used to create a dynamic configuration for MySQL, Nagios depend upon the resources.

**Example:**

MySQL innodb\_buffer\_pool have to be 70% of total RAM for better performance. So it’s easy to make it from ansible variables like,

mysql\_innodb\_buffer\_pool\_size: "{{ (ansible\_memtotal\_mb \* 0.7) | int }}M"

**Breakdown:**

ansible\_memtotal\_mb will be retrieved from the setup module. Basically, it will return the system stats and assigned it to respected variables.

Command to get complete stats about the system.

**To get stats about the local system:**

ansible --connection=local 127.0.0.1 -m setup

**To get stats about the remote system from the inventory file:**

ansible -i inventory\_file group\_name -m setup

This can be disabled by adding the “gather\_facts: no” in the respected host.

**Sample:**

- hosts: all

gather\_facts: no

Auto generated variable definitions using the ansible stats (system resources). Based on the condition it will revert the values for the respected variables.

Below is the sample yaml file which has the syntax and the variables.

**mysql\_conf.yml:**

---

# MySQL connection settings.

p: "3306"

mysql\_data\_dir: "/var/lib/mysql"

mysql\_pid\_file: "{{ mysql\_data\_dir }}/mysqld.pid"

mysql\_socket: "{{ mysql\_data\_dir }}/mysql.sock"

# Slow query log settings.

mysql\_slow\_query\_log\_enabled: yes

mysql\_slow\_query\_time: "2"

mysql\_slow\_query\_log\_file: "{{ mysql\_data\_dir }}/mysql-slow.log"

# Based on resources

mysql\_max\_connections: "{{ (ansible\_memtotal\_mb // 12) | int }}"

# Set ..\_buffer\_pool\_size up to 70% of RAM but beware of setting too high.

mysql\_innodb\_buffer\_pool\_size: "{{ (ansible\_memtotal\_mb \* 0.7) | int }}M"

# Set ..\_log\_file\_size to 25% of buffer pool size.

mysql\_innodb\_log\_file\_size: '{{ ((mysql\_innodb\_buffer\_pool\_size | string | replace("M", "") | int) \* 0.25) | int }}M'

example:

mysql\_port: "3306"

or

mysql\_port: "{{port}}"

mysql\_max\_connections: "10"

{{i}}|int

Int I

When we have the variable definition ready we need to apply it for generating the configuration file with required fields.

**mysql\_conf.j2: (template)**

# {{ ansible\_managed }}

[client]

port = {{ mysql\_port }}

socket = {{ mysql\_socket }}

[mysqld]

port = {{ mysql\_port }}

datadir = {{ mysql\_data\_dir }}

socket = {{ mysql\_socket }}

pid-file = {{ mysql\_pid\_file }}

# Slow query log configuration.

{% if mysql\_slow\_query\_log\_enabled %}

slow\_query\_log = 1

slow\_query\_log\_file = {{ mysql\_slow\_query\_log\_file }}

long\_query\_time = {{ mysql\_slow\_query\_time }}

{% endif %}

# InnoDB settings.

innodb\_buffer\_pool\_size = {{ mysql\_innodb\_buffer\_pool\_size }}

innodb\_log\_file\_size = {{ mysql\_innodb\_log\_file\_size }}

# Setting max connections

{% if mysql\_max\_connections | int > 3000 %}

max\_connections = 3000

thread\_cache\_size = {{ (3000 \* 0.15) | int }}

{% elif mysql\_max\_connections | int < 150 %}

max\_connections = 150

thread\_cache\_size = {{ (150 \* 0.15) | int }}

{% else %}

max\_connections = {{ mysql\_max\_connections }}

thread\_cache\_size = {{ (mysql\_max\_connections | int \* 0.15) | int }}

{% endif %}

Above will have the condition mapping along with the variable precedence. If the condition matches it will return the values with respect to the resource or it will keep the default value.

**Playbook:**

- hosts: 127.0.0.1

vars\_files:

- mysql\_conf.yml

tasks:

- name: Creating my.cnf with respected resources

template:

src: mysql\_conf.j2

dest: my.cnf

**Command to generate my.cnf using the template:**

ansible-playbook playbook.yml

**Output:**

Mydbops-MacBook-Air:jinja dhanasekar$ ansible-playbook playbook.yml

PLAY [127.0.0.1] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [127.0.0.1]

TASK [Creating my.cnf with respected resources] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [127.0.0.1]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0

my.cnf: (OUTPUT)

# Ansible managed

[client]

port = 3306

socket = /var/lib/mysql/mysql.sock

[mysqld]

port = 3306

datadir = /var/lib/mysql

socket = /var/lib/mysql/mysql.sock

pid-file = /var/lib/mysql/mysqld.pid

# Slow query log configuration.

slow\_query\_log = 1

slow\_query\_log\_file = /var/lib/mysql/mysql-slow.log

long\_query\_time = 2

# InnoDB settings.

innodb\_buffer\_pool\_size = 5734M

innodb\_log\_file\_size = 1433M

# Setting max connections

max\_connections = 682

thread\_cache\_size = 102

The above cnf was generated using the template. I hope it will give you a better idea about templating using Jinja2.

https://github.com/ansible/ansible-examples/tree/master/tomcat-standalone