IT Automation with

ANSIBLE

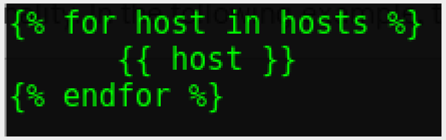
Agenda

- Jinja2 templates introduction
- Ansible Vault
- Debugging Ansible playbooks

Jinja2

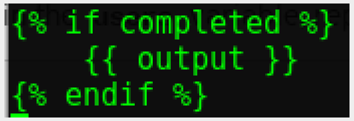
Loop and conditionals

- Jinja2 offers a loop structure which allows to iterate over a set of items



```
{% for host in hosts %}
      {{ host }}
{% endfor %}
```

- An **if** statement can be used against expressions such as



```
{% if completed %}
    {{ output }}
{% endif %}
```

Jinja2

Applying filters

- It is possible to change the value of expressions by applying filters to it. Jinja2 ships with many filters.
- Filters for formatting data:
  - JSON / YAML

```
{{ some_variable | to_json }}
{{ some_variable | to_yaml }}
```

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

- Alternatively formatted data may be read from variables:

```
{{ some_variable | from_json }}
{{ some_variable | from_yaml }}
```

```
tasks:
  - shell: cat /some/path/to/file.json
    register: result

  - set_fact:
      myvar: "{{ result.stdout | from_json }}"
```

Jinja2

Applying filters

- The default behavior of ansible is to fail if a variable is referenced and it has not been defined previously.
- Jinja2 allows to mark a variable as mandatory in the template by applying the **mandatory** filter as follows:

  ```
  {{ variable | mandatory }}
  ```

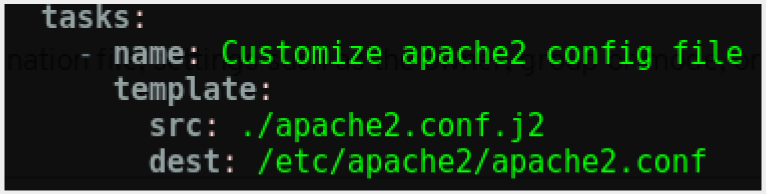
  In above example the variable value won’t be altered but will fail if the variable is undefined.
- Similarly a default filter exists in order to set values for undefined variables:

```
{{ some_variable | default(5) }}
```

Jinja2

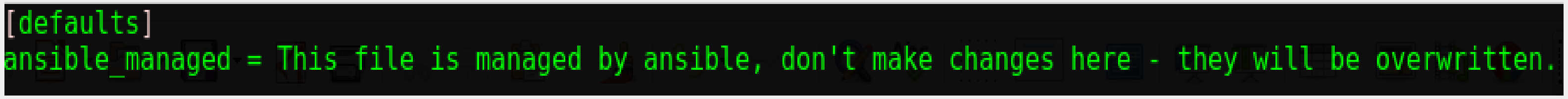
Loops

- Jinja2 templates are often used for configuration files through the template ***ansible*** module:



```
tasks:
  - name: Customize apache2 config file
    template:
      src: ./apache2.conf.j2
      dest: /etc/apache2/apache2.conf
```

- Usually template files will use **.j2** as extension.
- Although it is not strictly necessary it is recommended to include a ***{{ ansible_managed }}*** variable at the beginning of the template file to indicate this file is being managed by ansible.
- **Note**: for this to work, ansible_managed variable should be defined, e.g: in the ansible.cfg file:



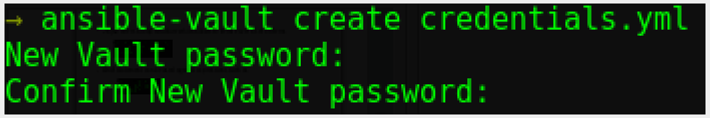
```
[defaults]
ansible_managed = This file is managed by ansible, don't make changes here - they will be overwritten.
```

Ansible Vault

Ansible Vault

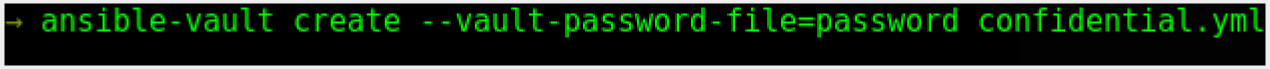
Basics

- Ansible Vault is a feature that allows you to encrypt sensitive fails, e.g: passwords or key files.
- Files are encrypted with AES 256 with a password as a symmetric key.
- Once the files containing sensitive information has been encrypted with Vault, then they can be set into a VCS.
- A command line tool ansible-vault exists to create, encrypt an existing file and change its key:
  - **ansible-vault create**:



```
→ ansible-vault create credentials.yml
New Vault password:
Confirm New Vault password:
```

Alternatively a password file can be used to set the vault password by using the --vault-password-file option.

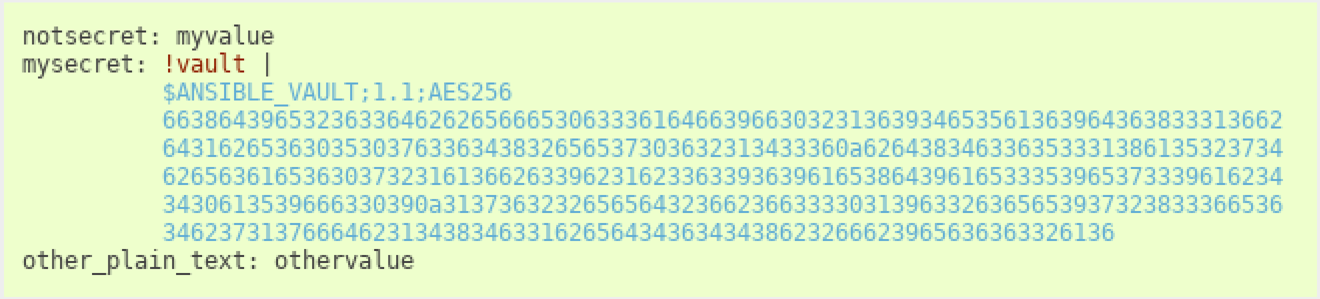


```
→ ansible-vault create --vault-password-file=password confidential.yml
```

Ansible Vault

Basics

- **Creating an encrypted file:** *ansible-vault create <file>* (the <file> must not exist).
- As of Ansible 2.3 it's possible to encrypt a single variable by using the ***ansible_vault encrypt_string*** command



```
notsecret: myvalue
mysecret: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66386439653236336462626566653063336164663966303231363934653561363964363833313662
          6431626536303530376336343832656537303632313433360a626438346336353331386135323734
          62656361653630373231613662633962316233633936396165386439616533353965373339616234
          3430613539666330390a313736323265656432366236633330313963326365653937323833366536
          34623731376664623134383463316265643436343438623266623965636363326136
other_plain_text: othervalue
```

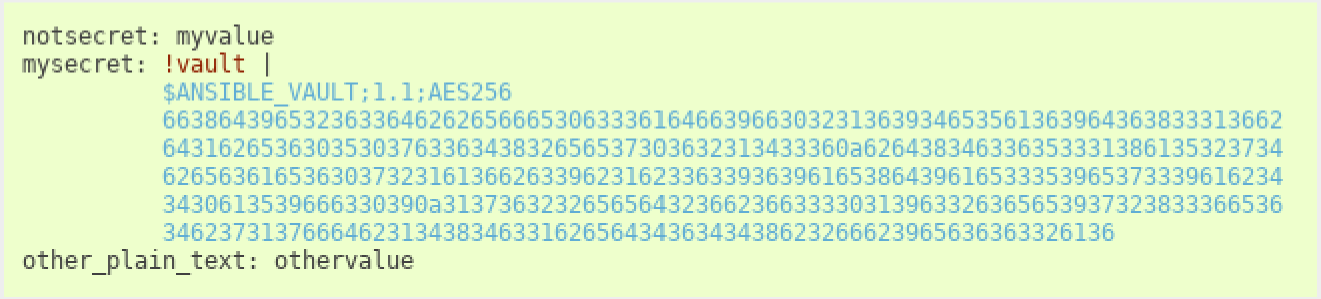
- **Edit encrypted files:** ansible-vault edit <file>

  Note: The edit option will always rewrite the file, this may have implications with VCS, therefore the view option should be used instead to see file's content.

Ansible Vault

Basics

- **Encrypting uncrypted file:** *ansible-vault encrypt <file>*
- **Decrypting encrypted file:** *ansible-vault decrypt <file>*



```
notsecret: myvalue
mysecret: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          66386439653236336462626566653063336164663966303231363934653561363964363833313662
          6431626536303530376336343832656537303632313433360a626438346336353331386135323734
          62656361653630373231613662633962316233633936396165386439616533353965373339616234
          3430613539666330390a313736323265656432366236633330313963326365653937323833366536
          34623731376664623134383463316265643436343438623266623965636363326136
other_plain_text: othervalue
```

- **Edit encrypted files:** *ansible-vault edit <file>*
- **Rekeying encrypted files:** *ansible-vault rekey <file>*

  Note: *a vault password file can be specified with --new-vault-password-file option*

Ansible Vault

How to use vault in playbooks

- To run a playbook that contains a vault-encrypted data files you must pass one of two flags:
  - **Specifying the vault password interactively:** *--ask-vault-password* will ask the user to provide the password to decrypt files that are accessed.

    Note: currently this requires that all files be encrypted with the same password.

    ```
    ansible-playbook site.yml --ask-vault-pass
    ```

  - **Passing a password file:** Passwords can also be specified through a password file by using the *--vault-password-file* option. In this case, the password should be stored as a single line in the file.

    ```
    ansible-playbook site.yml --vault-password-file ~/.vault_pass.txt

    ansible-playbook site.yml --vault-password-file ~/.vault_pass.py
    ```

Alternative ways to run playbooks

Debugging ansible playbooks

Basics

- By default Ansible does not keep a log of playbook execution, but it can be easily enabled by using either the **log_path** directive within *ansible.cfg* file or by defining the **$ANSIBLE_LOG_PATH** variable that should point to a log file.
- The **debug** module can also be useful in debugging playbooks misbehavior, e.g: by printing variable's value.
- There are some times where you might want to start a playbook from a certain task, this can be done by using the ***--start-at-task*** option to **ansible-playbook** command:

```
ansible-playbook playbook.yml --start-at-task="install packages"
```

  Basically this will skip all previous tasks in the playbook execution.

- Similarly there may be situations where you may want to run a playbook step by step, i.e: making Ansible stop on each task and ask whether you want to execute next task. This is possible by passing the option ***--step*** to **ansible-playbook:**

```
ansible-playbook playbook.yml --step
```

```
Perform task: configure ssh (y/n/c):
```

Debugging ansible playbooks

Basics

- Lastly, there might be times where you may need more information about a certain error and the information got by just running the playbook is not enough. In these cases you may want to increase verbosity level in playbook execution.
- There are 4 levels of verbosity in total, being -v the default one and -vvvv the most detailed one:

Option	Description
-v	The default level. Module output is shown
-vv	Apart from the module's output, input data is also shown.
-vvv	Includes information about network connection to managed hosts
-vvvv	Adds verbosity for connection plugins, and for scripts that are executed. Mostly used when connection debugging is needed