

TEMPLATING & VARIABLES

DÉCLARER UNE VARIABLE

- Les variables peuvent être définies à de multiples niveaux au sein d'Ansible

- **Au niveau de la ligne de commande** via l'argument `-e` ou `--extra-vars`



Code : Argument extra var

```
-e "var1=var1_value var2=var2_value"  
-e '{"var1":"var1_value","var2":"var2_value"}'  
-e '{"var1":"var1_value","var2":["var2_value1","var2_value2"]}'  
-e "@my_file.json"
```

- **Au niveau des scripts ansible**

- playbooks ;
- inventaires – groupes, sous groupes, nœuds ;
- rôles ;
- tâches ;



La déclaration des variables au sein des playbooks, inventaires, rôles et tâches est précisée au sein du § de chaque notion

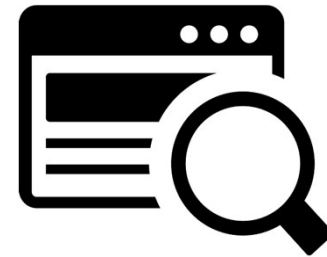
- **Via les faits relatifs aux nœuds managés** (cf. fact gathering)



TEMPLATING & VARIABLES

DÉFINIR UN TEMPLATE

- Les templates permettent de produire des fichiers dont le contenu est valorisé par des variables au moment de l'exécution des tâches ansible
- Ansible offre différentes façons de gérer le templating :
 - Via le module **lineinfile**
 - Via le module **replace**
 - Via le module **blockinfile**
 - Via le module **template** en utilisant le langage de templating **jinja2**



TEMPLATING & VARIABLES

MODULE LINE IN FILE (1/2)

- le module **lineinfile** permet remplacer **la ligne** d'un fichier en s'appuyant sur **une expression régulière**



Code : lineinfile au sein d'un rôle

```
- name:      Ensure SSH access by group is enabled
  become:    yes
  become_method: sudo
  lineinfile:
    dest:     /etc/ssh/sshd_config
    regexp:   "^AllowGroups"
    line:     "AllowGroups {{security_ssh__group_name}}"
    state:    present
    backup:   yes
```

Valeur de la variable `security_ssh__group_name`



TIP Pour tester vos expressions régulières ?
<https://regex101.com/>



state = present : si plusieurs lignes matchent le pattern de l'expression régulière, **seule la dernière** sera remplacée

state = absent : si plusieurs lignes matchent le pattern, **toutes les lignes** seront supprimées



TEMPLATING & VARIABLES

MODULE LINE IN FILE (2/2)



A noter : les variables peuvent aussi être utilisées dans les noms des tâches



Code : lineinfile au sein d'un rôle avec **validation** de la modification

```
- name:      Add group "{{sudo__passwordless_group_name}}" to sudoers file
  become:    yes
  become_method: sudo
  lineinfile:
    dest:     /etc/sudoers
    regexp:   '^{{sudo__passwordless_group_name}}'
    line:     '{{sudo__passwordless_group_name}} ALL=(ALL:ALL) NOPASSWD:ALL'
    state:    present
    backup:   true
    validate: visudo -cf %s
```

Lance une commande de validation avant modification effective
%s correspond au fichier à valider



Si la regexp ne trouve **aucun résultat** et que le **state est présent**, la ligne sera rajoutée en fin de fichier (par défaut) ou selon la valeur de insertbefore ou insertafter



TEMPLATING & VARIABLES

MODULE REPLACE

- Le module **replace** permet de remplacer **toutes les valeurs** respectant un pattern au sein d'un fichier en s'appuyant sur une **expression régulière**



Code : replace au sein d'un rôle

```
- name: Commente le contenu du bloc virtualhost
  replace:
    path: /etc/httpd/...
    after: '<VirtualHost [*]>'
    before: '</VirtualHost>'
    regexp: '^(.+)$'
    replace: '# \1'
```

() : groupe de capture

\1 : le contenu du groupe de capture



Certains caractères spéciaux pour les regexp – `()[].*+ –` doivent être échappés au sein de la regexp avec `\`



TEMPLATING & VARIABLES

MODULE BLOCK IN FILE

- Le module **blockinfile** permet de remplacer **un bloc de ligne** d'un fichier en s'appuyant sur une **expression régulière**



Code : *blockinfile* au sein d'un rôle

```
- name: Insert/Update HTML surrounded by custom markers after <body> line
  blockinfile:
    path: /var/www/html/index.html
    marker: "<!-- {mark} ANSIBLE MANAGED BLOCK -->"
    insertafter: "<body>"
    block: |
      <h1>Welcome to {{ ansible_hostname }}</h1>
      <p>Last updated on {{ ansible_date_time.iso8601 }}</p>
```



A noter : l'opérateur multiligne `'|'`
plusieurs opérateurs YAML existent :
| remplace les sauts de ligne par des espaces
> conserve les sauts de ligne



TEMPLATING & VARIABLES

MODULE BLOCK IN FILE (2/2)

notions
avancées



Code : *blockinfile* avec loop

```
- name: Add mappings to /etc/hosts
blockinfile:
  path: /etc/hosts
  marker: "# {mark} ANSIBLE MANAGED BLOCK"
  block: |
    {{ item.ip }} {{ item.name }}
  loop:
    - { name: host1, ip: 10.10.1.10 }
    - { name: host2, ip: 10.10.1.11 }
    - { name: host3, ip: 10.10.1.12 }
```



Quand utiliser un template jinja2 versus blockinfile ?

Si vous avez la totale maîtrise du fichier cible, privilégier un template jinja2



TEMPLATING & VARIABLES

MODULE TEMPLATE (JINJA2)



- Le module **template** en utilisant le langage de templating **jinja2**
 - Permet de valoriser plusieurs variables au sein du template
 - Permet de s'appuyer sur des structures de contrôle élaborées (boucle, condition, ...)



Code : template au sein d'un rôle + validation

```
- name: template /etc/nginx/nginx.conf with validation
  become: true
  template:
    src: templates/etc/nginx/nginx.conf.j2
    dest: /etc/nginx/nginx.conf
    owner: root
    group: root
    mode: 0644
    validate: /usr/bin/nginx -t -c %s
```







A noter : il est préférable de mettre les fichiers de templating dans le répertoire templates du rôle concerné et de conserver le chemin cible

*Lance une commande de validation avant modification effective
%s correspond au fichier à valider*



TEMPLATING & VARIABLES

FICHER JINJA2

Commande	Description	Exemple
<code>{# mon commentaire #}</code>	commentaires qui ne seront pas intégrés à la sortie produite	
<code>{{ variable }}</code>	la valeur de la variable	
<code>{% ... %}</code>	à utiliser pour les structures de contrôle (type boucles ou conditions)	<div>Code : condition jinja2<pre>{% if variable is defined %} value of variable: {{ variable }} {% else %} variable is not defined {% endif %}</pre></div> <div>Code : boucle jinja2<pre>{% for i in web_servers %} nom du serveur {{ i }} {% endfor %}</pre><div>web_servers est une variable</div></div>
<code>{{ variable filtre1 }}</code>	Application d'un 'filtre' jinja2 à la variable	<div>Code : filtre<pre>{{ variable upper }}</pre></div>
<code>{{ variable filtre1 filtre2 filtre3 }}</code>	Chainage de filtres via pipping (pour les dev, équivalent à DSL <code>filtre1(variable).filtre2().filtre3()</code>)	<div>Code : filtres + pipping jinja2<pre>{{ variable default("non définie") upper }}</pre></div>



TEMPLATING & VARIABLES

PRINCIPAUX FILTRES JINJA2 (1/2)

Filtre	Description
default()	pour affecter une valeur par défaut si variable absente
join()	pour joindre une liste avec un séparateur
dirname / basename	pour extraire des portions d'un nom de fichier
bool, int, float	pour forcer le cast d'un objet
replace()	pour faire une substitution dans une chaîne
sort	pour trier une liste
upper / lower	pour changer la casse
union / intersect / difference	pour manipuler des listes
regex_replace	remplacements à base d'expressions régulières
match	vérification qu'une chaîne respecte un pattern donné
select	Sélectionne les objets d'une liste qui vérifie le filter
selectattr	Sélectionne les objets d'une liste dont l'attribut vérifie le filtre



TEMPLATING & VARIABLES

PRINCIPAUX FILTRES JINJA2 (2/2)



Code : filtre hash et checksum

```
sha1_hash: "{{ 'test1' | hash('sha1') }}"
checksum: "{{ 'test1' | checksum }}"
```



Code : pipelining

```
users:
  - name: john
    email: john@example.com
  - name: jane
    email: jane@example.com
  - name: fred
    email: fred@example.com
    password: 123!abc

- set_fact:
  emails: "{{ users | selectattr('password', 'undefined') | map(attribute='email') | list }}"
```

notions
avancées

2 On filtre sur les utilisateurs n'ayant pas de password

4 On restitue ces emails sous forme de liste stockée dans la variable emails

1 On prend en données sources les utilisateurs

3 On ne conserve que l'email des items filtrés



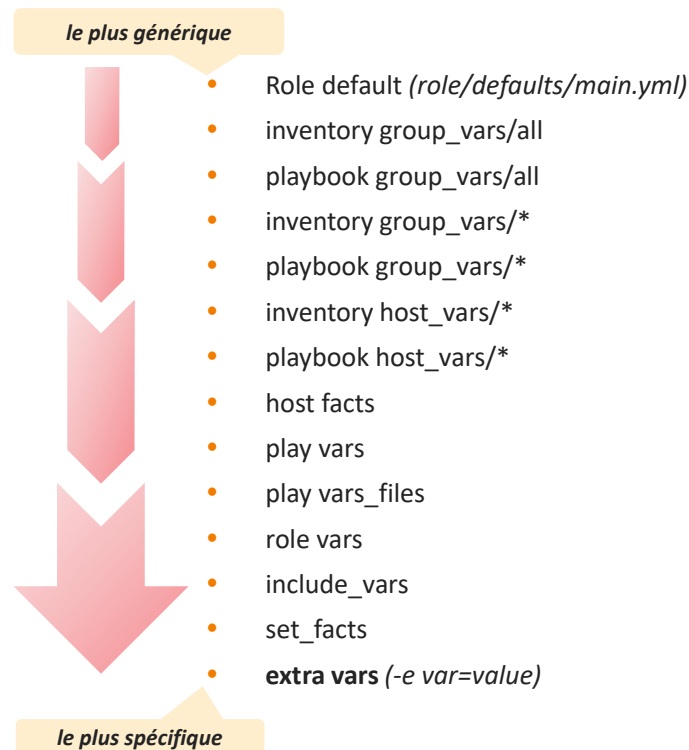
Le pipelining demande de l'habitude et viendra avec l'expérience!



TEMPLATING & VARIABLES

REGLE DE PRECEDENCE DES VARIABLES

- Ansible permet de **déclarer des variables à de multiples endroits**, ce qui peut provoquer un **risque de collision** sur le nom des variables.
- Dans le cas où une variable est déclarée à différents endroits, la **règle de précedence** ci-dessous est appliquée.



Règle de précedence

Le plus spécifique l'emporte sur le plus générique



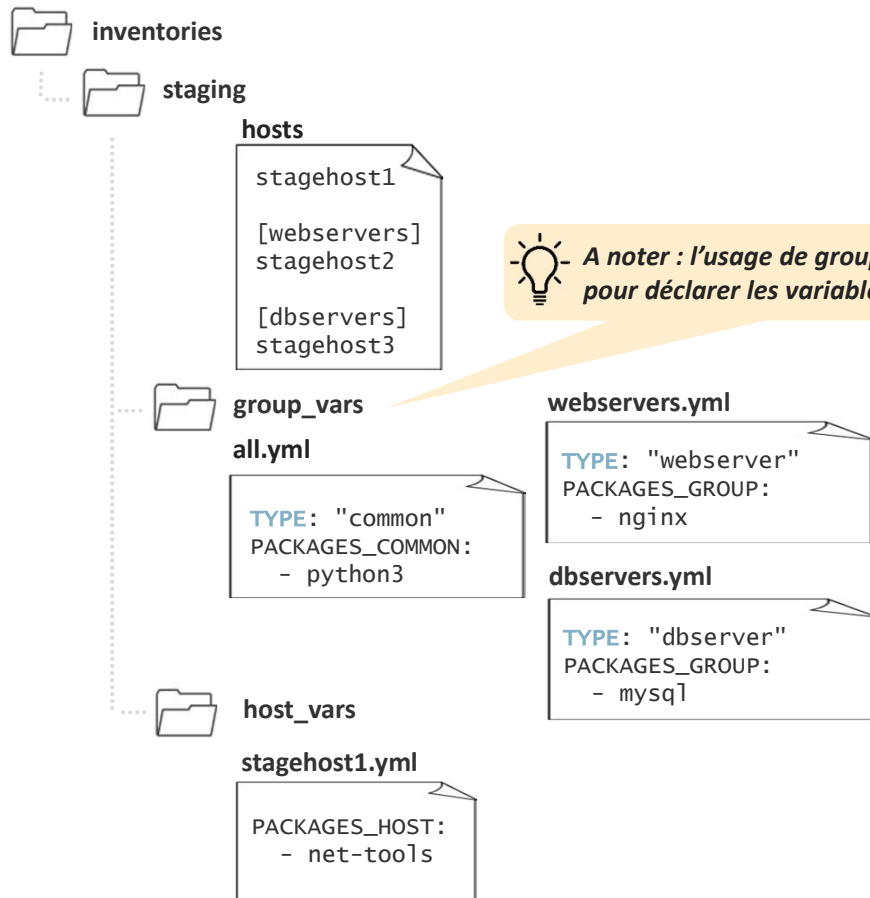
Pour éviter les collisions sur les variables?

Ajouter le nom du rôle comme préfixe au nom de la variable



TEMPLATING & VARIABLES

PRECEDENCE DES VARIABLES : EXEMPLE



A noter : l'usage de `group_vars` et `host_vars` pour déclarer les variables d'inventaire



Code : Débuguer les variables d'un inventaire

```
$ ansible-inventory -i inventories/staging --graph --vars
```

```
@all:
  |--@dbserver:
  |   |--stagehost3
  |   |   |--{PACKAGES_COMMON = ['python3']}
  |   |   |--{PACKAGES_GROUP = ['mysql']}
  |   |   |--{TYPE = dbserver}
  |   |--{PACKAGES_GROUP = ['mysql']}
  |   |--{TYPE = dbserver}
  |--@ungrouped:
  |   |--stagehost1
  |   |   |--{PACKAGES_COMMON = ['python3']}
  |   |   |--{PACKAGES_HOST = ['net-tools']}
  |   |   |--{TYPE = common}
  |--@webserver:
  |   |--stagehost2
  |   |   |--{PACKAGES_COMMON = ['python3']}
  |   |   |--{PACKAGES_GROUP = ['nginx']}
  |   |   |--{TYPE = webserver}
  |   |--{PACKAGES_GROUP = ['nginx']}
  |   |--{TYPE = webserver}
  |--{PACKAGES_COMMON = ['python3']}
  |--{TYPE = common}
```



Code : Afficher les variables

```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: append packages
      set_fact:
        PACKAGES: '{{PACKAGES_HOST | default ([]) + PACKAGES_COMMON + PACKAGES_GROUP}}'
    - name: debug
      debug:
        var: PACKAGES, TYPE
```

```
ok: [stagehost1] => {
  "PACKAGES,TYPE": "(['net-tools', 'python3'], 'common')"
}
ok: [stagehost3] => {
  "PACKAGES,TYPE": "(['mysql', 'python3'], 'dbserver')"
}
ok: [stagehost2] => {
  "PACKAGES,TYPE": "(['nginx', 'python3'], 'webserver')"
```



RÉCAPITULATIF

Comment définit-on les variables ?

Comment utilise-t-on les variables ?

A quoi sert la surcharge de variable?

Comment s'appelle le langage de templating?

Que sont les templates ?

Comment utilise-t-on les templates ?

