



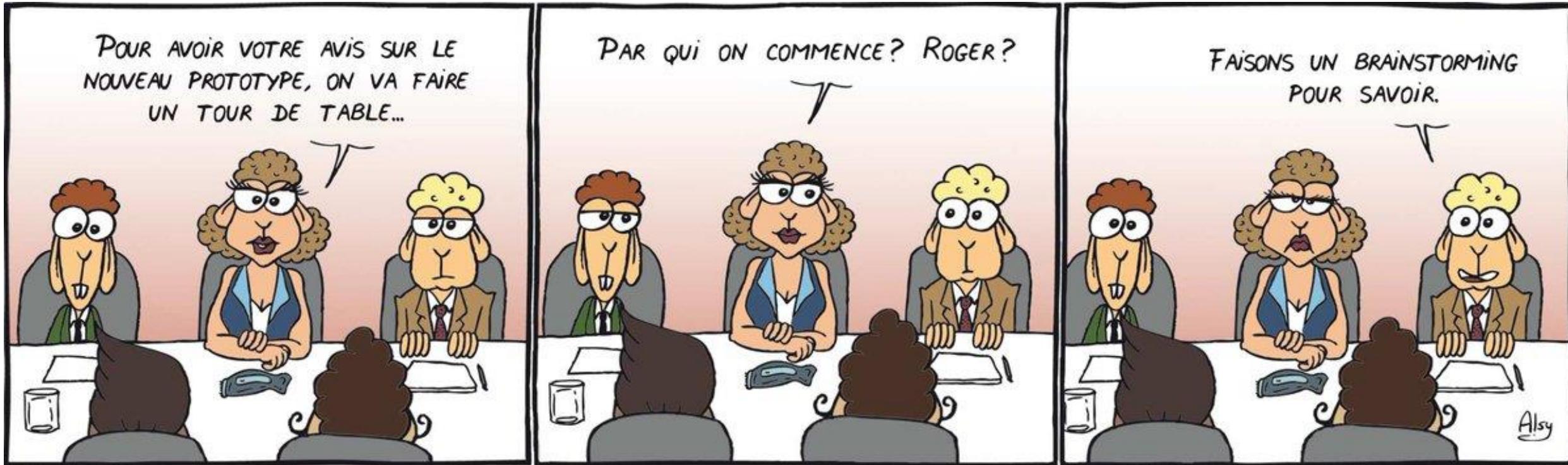
ANSIBLE – INITIATION

Version 1.3.2

Delivering Transformation. Together.*

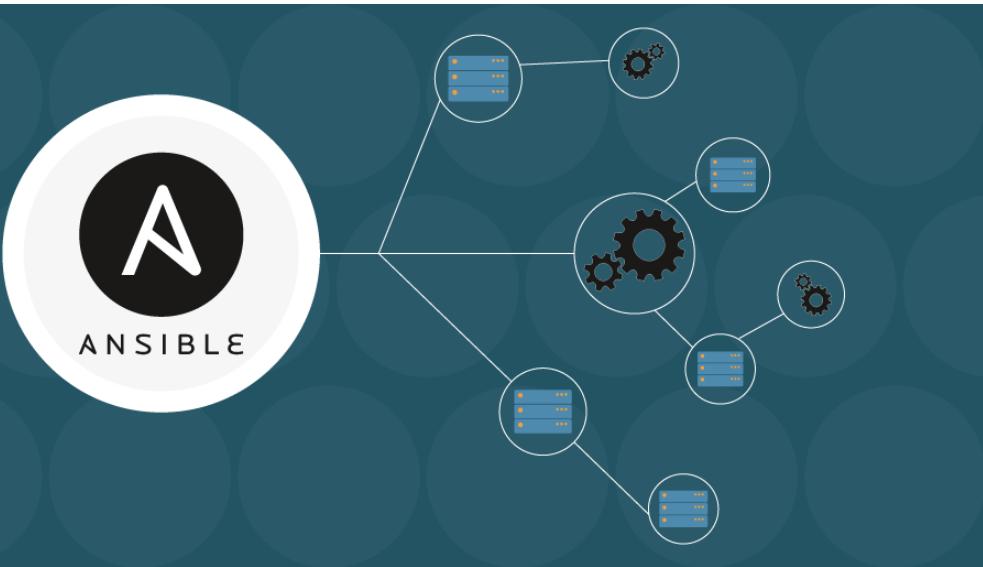
sopra  steria

TOUR DE TABLE



https://twitter.com/alsy_lesmoutons/status/915257954735026176

SOMMAIRE



jour 1	01	Introduction - Philosophie
	02	Inventaires - Modules - Playbook
	03	Notify/Handler - Conditionner
jour 2	04	Rôles
	05	Templating et Variables
	06	Avancé
	07	Vault
jour 3	08	Tests Unitaires
	09	Bonnes Pratiques & Optimisation



INTRODUCTION

DÉFINITION WIKIPEDIA



ANSIBLE

Ansible est une **plate-forme logicielle libre** pour la **configuration** et la **gestion** des ordinateurs. Elle combine le **déploiement** de logiciels (en) **multi-nœuds**, l'exécution des **tâches ad-hoc**, et la gestion de configuration.



INTRODUCTION

ANSIBLE C'EST...

- Un outil de gestion de configuration

IMPORTANT

- On définit un **état cible**
- Ansible
 - Calcule les actions nécessaires pour atteindre l'état cible
 - Applique les actions nécessaires

Nous y reviendrons plus loin...

- Un outil de déploiement automatisé

- **Orchestre** les différentes étapes d'un déploiement applicatif

- Un outil **sans agent**

- Racheté par RedHat le 16/10/2015



INTRODUCTION

ANSIBLE, AUTOMATISATION DES DÉPLOIEMENTS

Pourquoi automatiser les déploiements ?

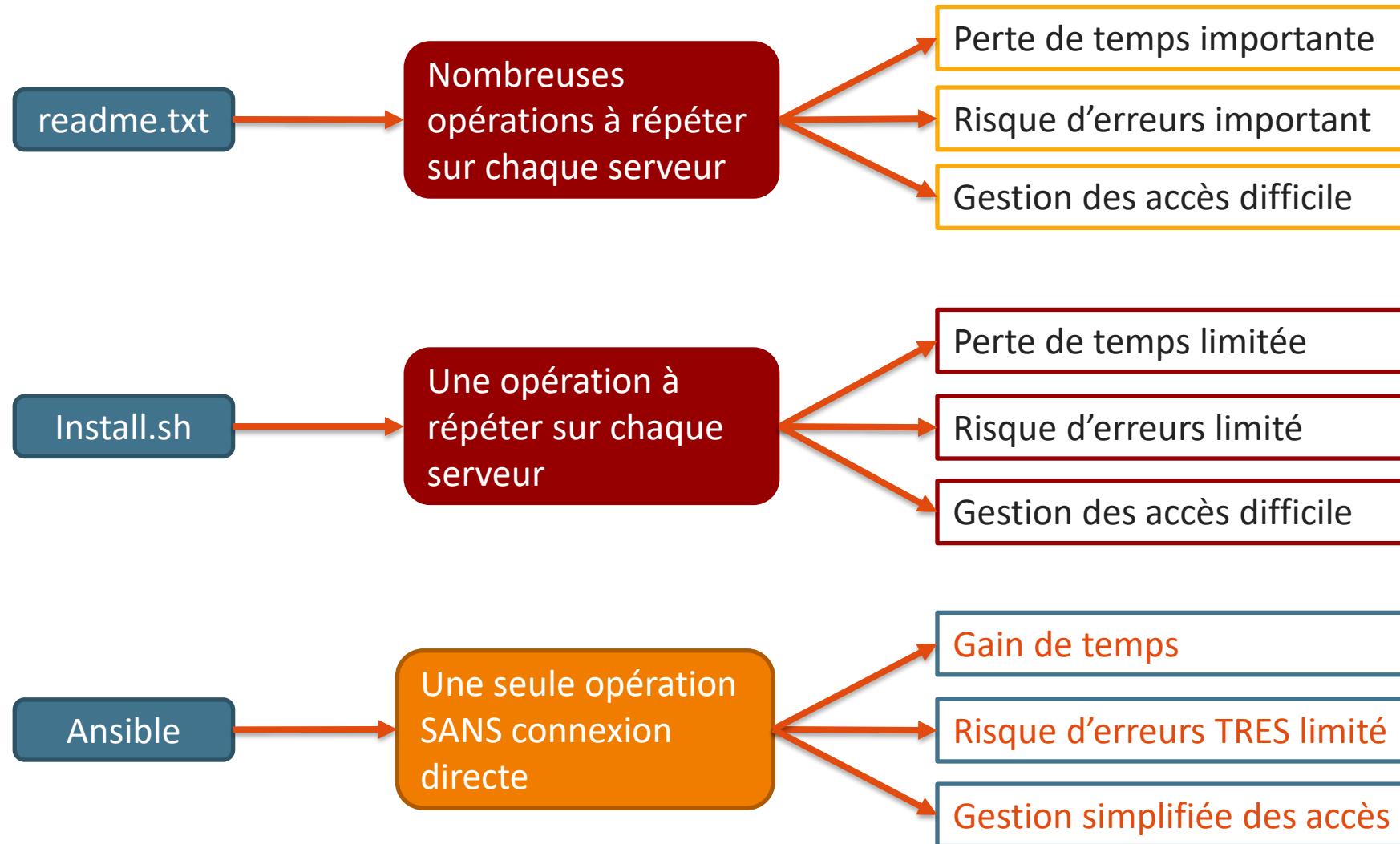
Différences entre

- le guide d'installation ([readme.txt](#))
- le script d'installation ([install.sh](#))
- le déploiement automatique ([Ansible](#))



INTRODUCTION

ANSIBLE, AUTOMATISATION DES DÉPLOIEMENTS



INTRODUCTION

ANSIBLE, PERMET DE...

- Gérer les configurations des machines
 - Installer des logiciels
 - Gérer des configurations (fichiers, services, ...)
- Orchestrer des déploiements d'applications
 - Potentiellement complexes
 - En synchronisant les actions entre plusieurs machines
- Provisionner des environnements
 - IaaS / Cloud : AWS, Azure, Google Compute Engine, OpenStack, VMWare...
- Piloter des appliances
 - F5 Big IP, Nagios,



*On ne définit pas « comment »
faire les choses*



**On définit « l'état » dans lequel
on veut que les choses soient**



PHILOSOPHIE

PROGRAMMATION PAR ÉTAT (1/2)

MODE FONCTIONNEL



Code : exemple d'installation d'apache par ligne de commande

```
$ apt-get install apache
```



Je décris **COMMENT** installer Apache



Dans cette ligne de commande, le binaire apt tentera d'installer apache même si apache était **DÉJÀ** installé !

PHILOSOPHIE

PROGRAMMATION PAR ÉTAT (2/2)

MODE PAR ETAT



Code : exemple d'installation d'apache par script Ansible

```
apt:  
    name: apache  
    state: present
```



Je décris **dans quel état** doit être apache

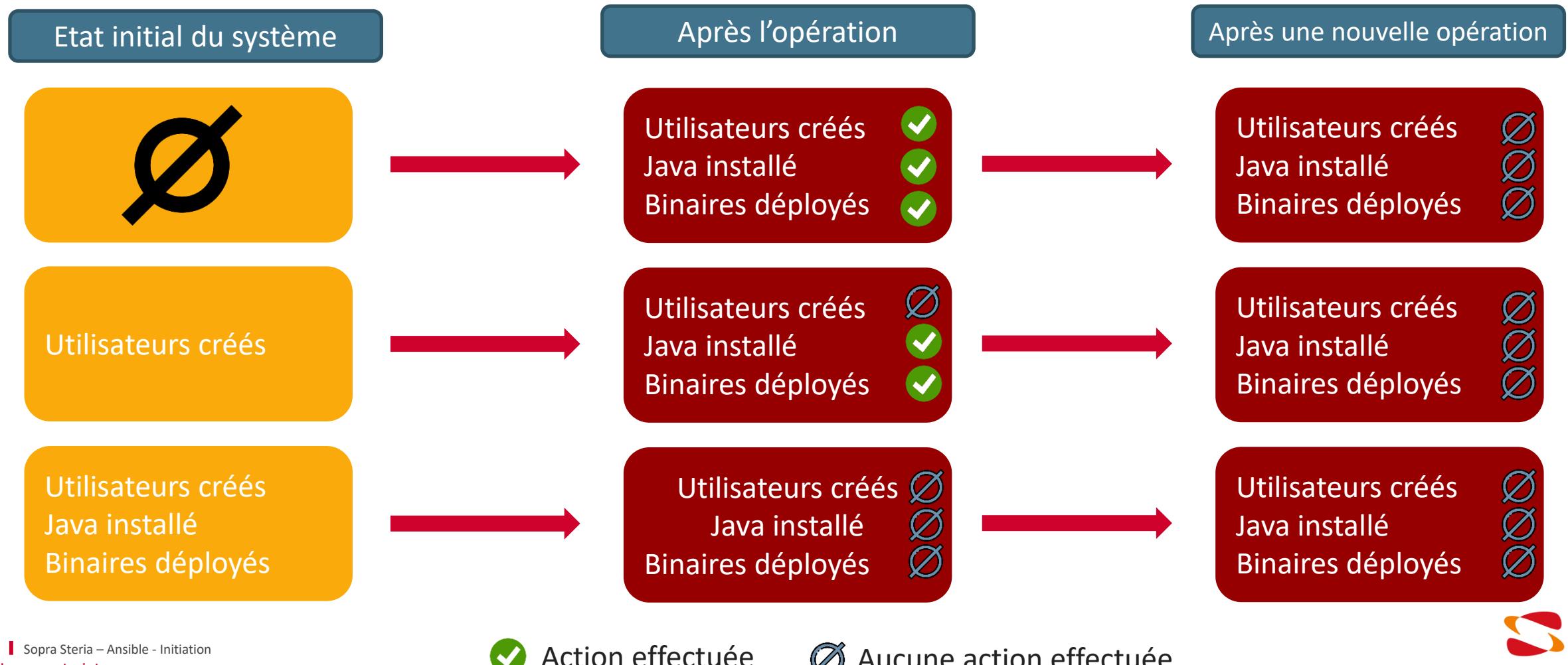


Dans cet extrait de script Ansible, si apache est déjà installé, Ansible ne fera aucune action



PHILOSOPHIE IDEMPOTENCE

Définition : Propriété d'une opération, d'avoir le même effet qu'on l'applique une ou plusieurs fois.



PHILOSOPHIE

ANSIBLE, AGENT-LESS

Communication entre le **serveur de déploiement** et
les nœuds à installer/configurer

Connexion SSH

Pas d'agent

Un seul port



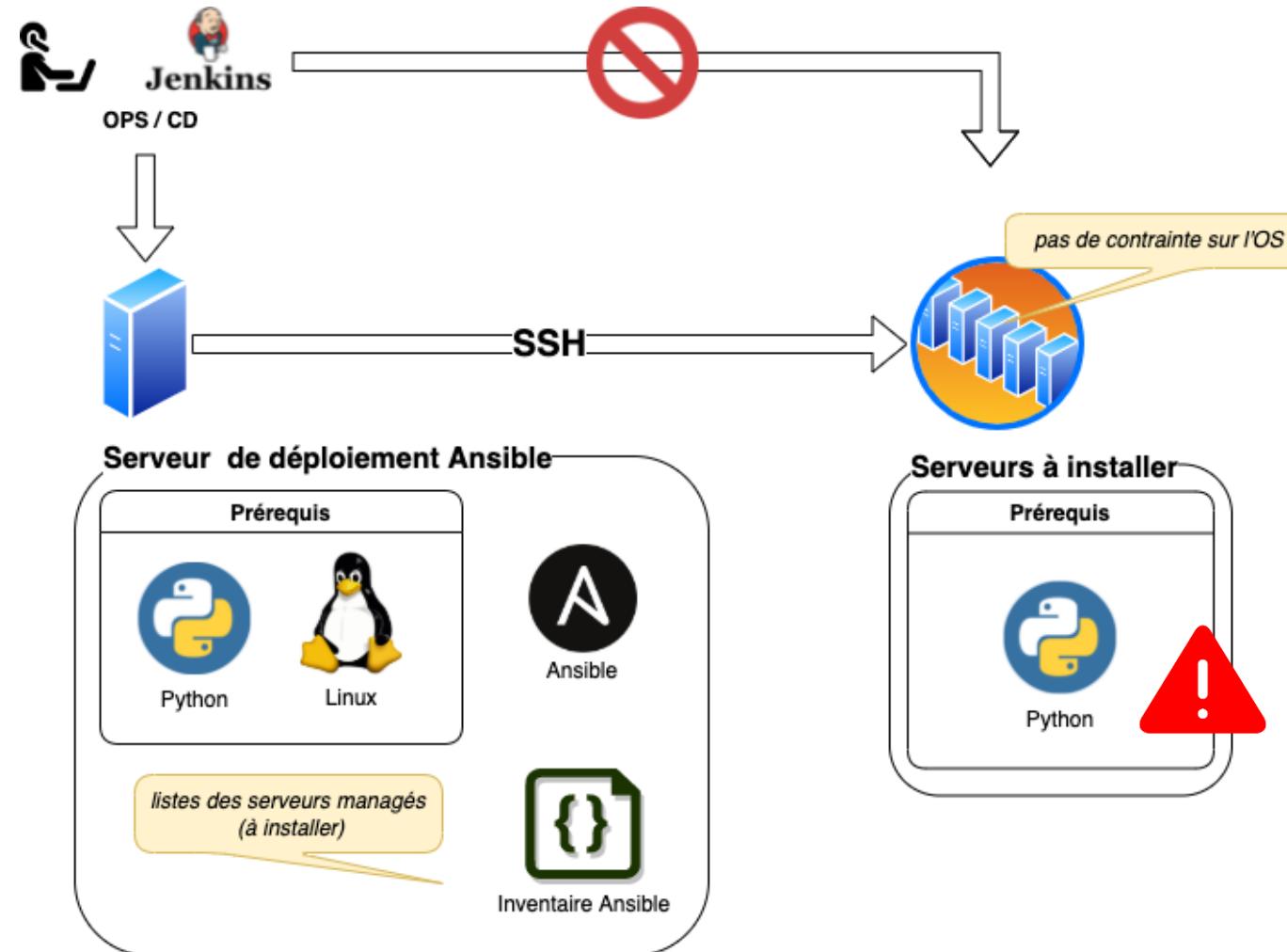
Aucune modification à apporter sur les serveurs à installer **

** nécessite python sur les serveurs pour traiter la sortie des commandes



PHILOSOPHIE

ANSIBLE SE CONNECTE EN SSH (1/2)



Sécurisation des accès

Centralisation & simplification des droits d'accès

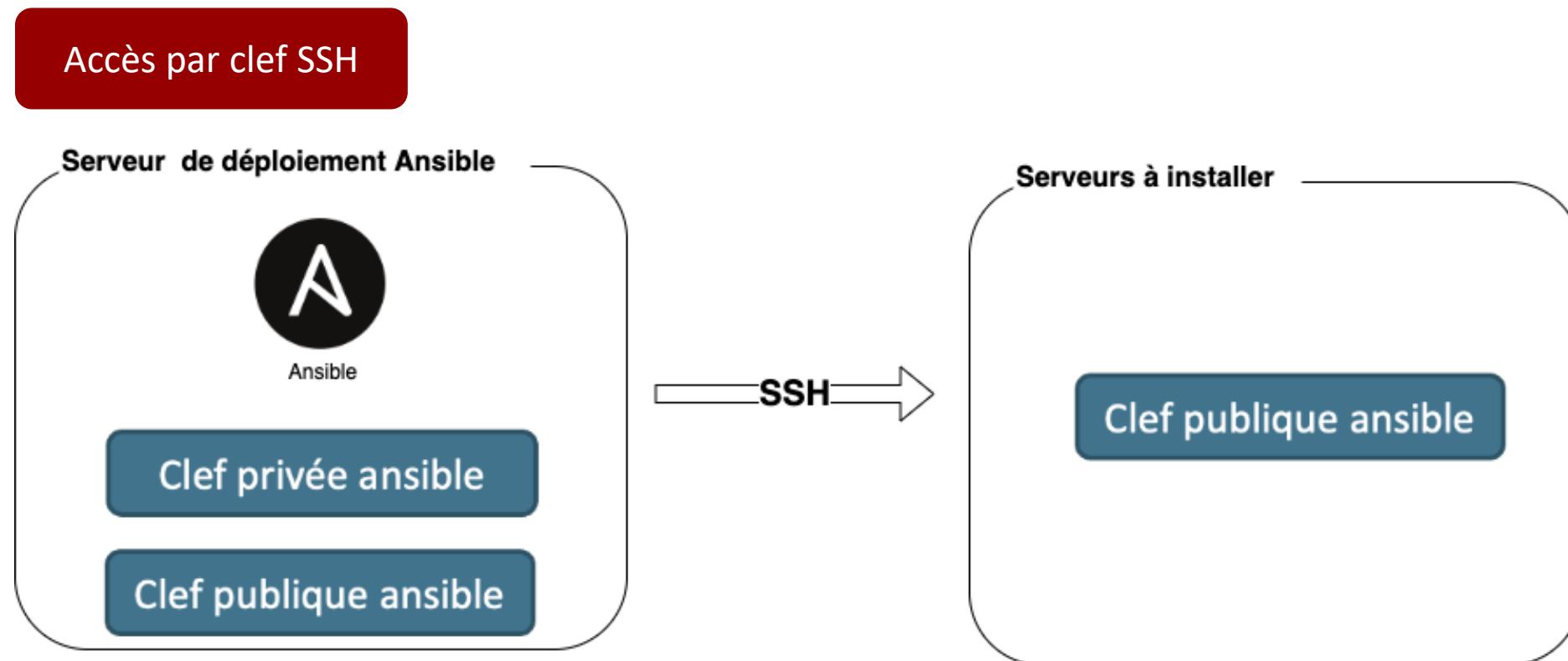
Simplification des flux

Certains modules utilisés par Ansible peuvent nécessiter une version minimale de python sur le serveur à installer



PHILOSOPHIE

ANSIBLE SE CONNECTE EN SSH (2/2)



Connexion SSH en root ou en user avec une élévation de privilèges via sudoers



ANSIBLE

LA JUNGLE DES VERSIONS

A juillet 2023

Ansible Community Package	Ansible Core	Statut	Changelog
7.0.0	2.14	dev	
6.x	2.13	courant	https://github.com/ansible-community/ansible-build-data/blob/main/6/CHANGELOG-v6.rst
5.x	2.12	EOL < 5.10	https://github.com/ansible-community/ansible-build-data/blob/main/5/CHANGELOG-v5.rst
4.x	2.11	EOL	https://github.com/ansible-community/ansible-build-data/blob/main/4/CHANGELOG-v4.rst
3.x	2.10	EOL	https://github.com/ansible-community/ansible-build-data/blob/main/3/CHANGELOG-v3.rst
2.10	2.10	EOL	https://github.com/ansible-community/ansible-build-data/blob/main/2.10/CHANGELOG-v2.10.rst
NA	2.9	EOL	

Ansible Community Package intègre plus des 85 collections (librairies).

Ansible Core permet de sélectionner les collections uniquement nécessaires.

Les montées de version apportent principalement des mises à niveau sur les versions de modules détaillés dans les changelogs.



Pour aller plus loin : https://docs.ansible.com/ansible/latest/reference_appendices/release_and_maintenance.html

ANSIBLE CONFIGURATION (1/2)

- Ansible est configurable via un fichier de configuration (par défaut /etc/ansible/ansible.cfg)
- Le fichier de configuration permet de modifier le fonctionnement et les fonctionnalités d'Ansible.
 - L'utilisateur par défaut avec lequel ansible se connecte aux serveurs
 - La méthode utilisée pour l'élévation des privilèges
 - Le répertoire par défaut des rôles
 - Les options de connexion SSH par défaut
 - La verbosité
 - etc



Code : Sections du fichier de config

```
[defaults]
[inventory]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[persistent_connection]
[accelerate]
[selinux]
[colors]
[diff]
```



Code : Section default

```
[defaults]

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml

forks          = 5
#poll_interval = 15

sudo_user      = root
#ask_sudo_pass = True
#ask_pass       = True
#transport     = smart
remote_port    = 345
#module_lang   = C
#module_set_locale = False
```



Pour aller plus loin : https://docs.ansible.com/ansible/latest/reference_appendices/config.html

ANSIBLE CONFIGURATION (2/2)

/etc/ansible/ansible.cfg

Fichier de configuration global d'Ansible. Ce fichier est utilisé par défaut

Fichier ansible.cfg local

Fichier dans le même répertoire que le script ansible.
Ce fichier surcharge le fichier de configuration global.

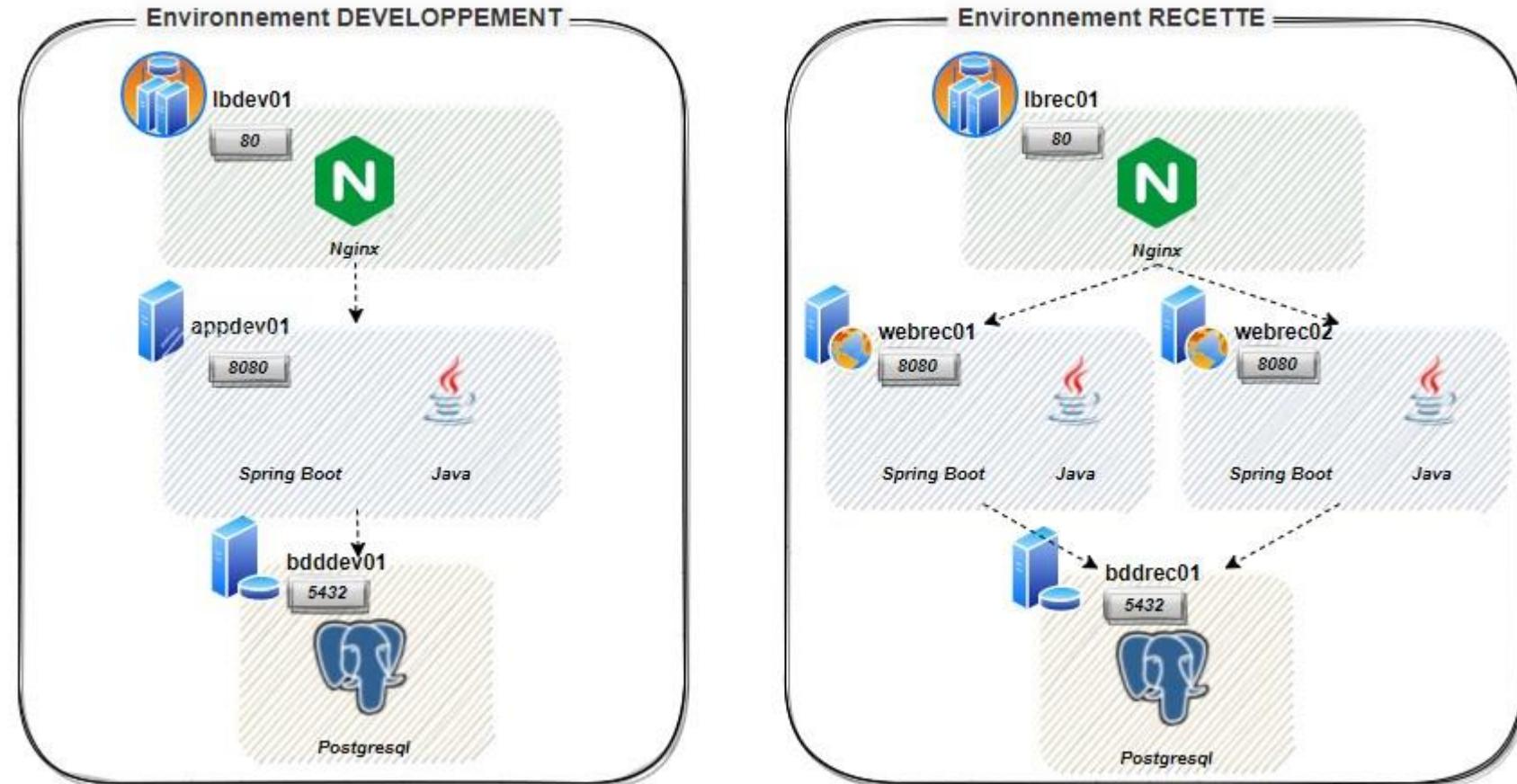


FIL ROUGE DU TP PRESENTATION

- Les différentes notions abordées durant la formation seront mises en pratique via l'écosystème cible ci-dessous :

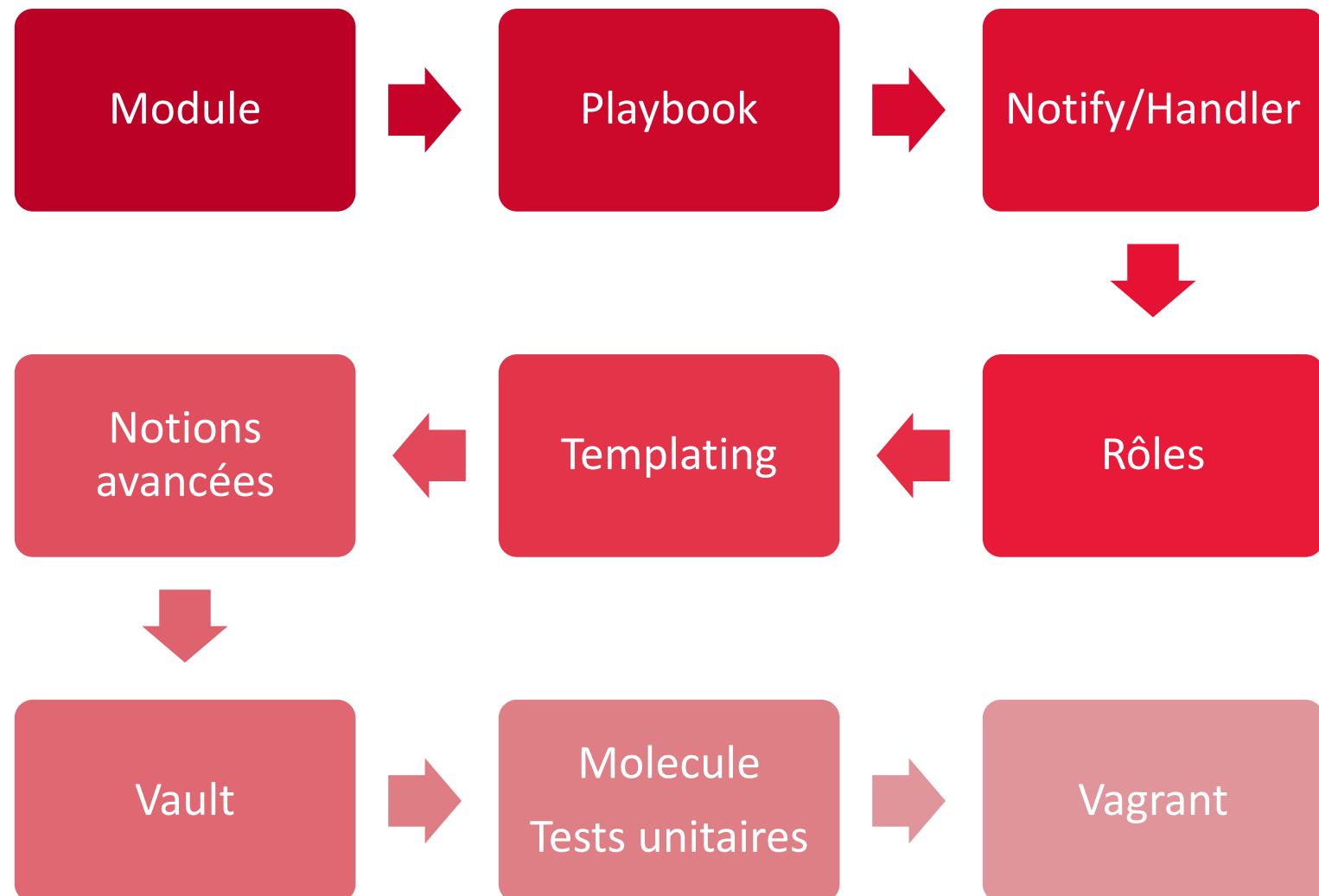
Java : open-jdk 17
Postgresql : 15

PetClinic



FIL ROUGE DU TP

LES DIFFÉRENTES ÉTAPES DES TP



FIL ROUGE DU TP

MISE EN PLACE ENVIRONNEMENT (1/3)

Se connecter sur l'URL : <https://fabricator.ansible.cloud-sp.eu/envs/Ansible>

The screenshot shows the Ansible Fabricator interface. At the top, there's a navigation bar with 'Environnements' and 'Home'. On the right, an email address 'alexandre.blanc@sopraSterianext.com' is displayed. A red arrow points from the text 'Votre mail de connexion' to this email address. Below the navigation bar, the title 'Ansible' is followed by 'Les environnements existants'. A table lists environments: one row for 'alexandre.blanc' with three URLs: 'alexandre-blanc-ansible.ansible.cloud-sp.eu', 'lbdev01-alexandre-blanc-ansible.ansible.cloud-sp.eu', and 'lbreco1-alexandre-blanc-ansible.ansible.cloud-sp.eu'. Another row shows 'Status' as 'Active' and 'Clean' status as 'ALL', 'DEV', and 'REC'. A red box encloses the URLs and another encloses the 'Clean' status buttons. Red arrows point from the text 'Accéder à votre environnement de travail' to the URLs and from 'Remettre à zéro une partie ou la totalité des VMs des TPs' to the 'Clean' status buttons.

Name	Url	Status	Clean
alexandre.blanc	alexandre-blanc-ansible.ansible.cloud-sp.eu lbdev01-alexandre-blanc-ansible.ansible.cloud-sp.eu lbreco1-alexandre-blanc-ansible.ansible.cloud-sp.eu	Active	ALL DEV REC

Votre mail de connexion

alexandre.blanc@sopraSterianext.com

Environnements Home

Ansible

Les environnements existants

Name	Url	Status	Clean
alexandre.blanc	alexandre-blanc-ansible.ansible.cloud-sp.eu lbdev01-alexandre-blanc-ansible.ansible.cloud-sp.eu lbreco1-alexandre-blanc-ansible.ansible.cloud-sp.eu	Active	ALL DEV REC

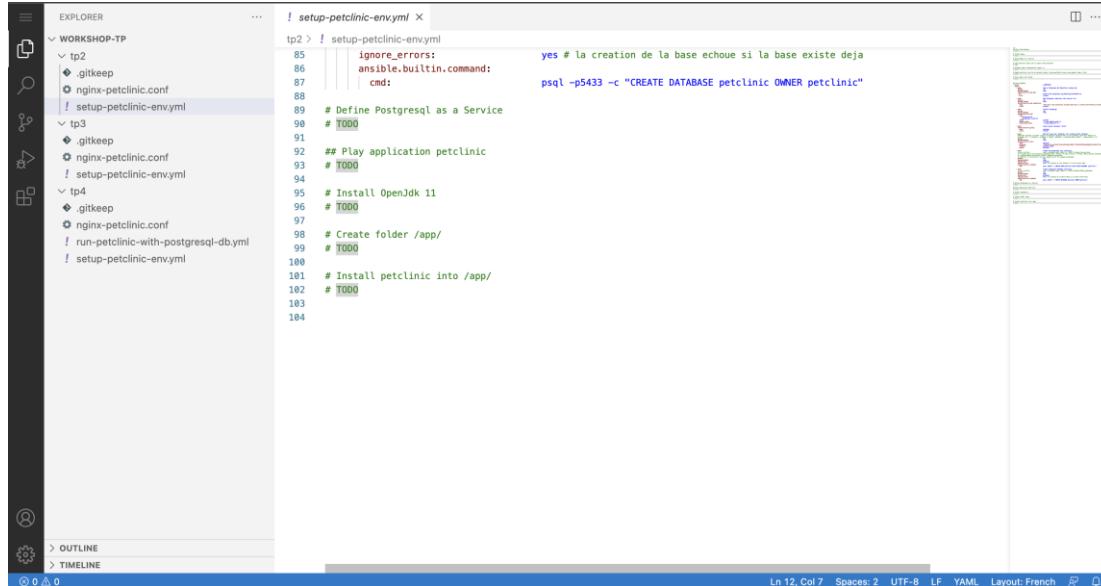
Accéder à votre environnement de travail

Remettre à zéro une partie ou la totalité des VMs des TPs



FIL ROUGE DU TP

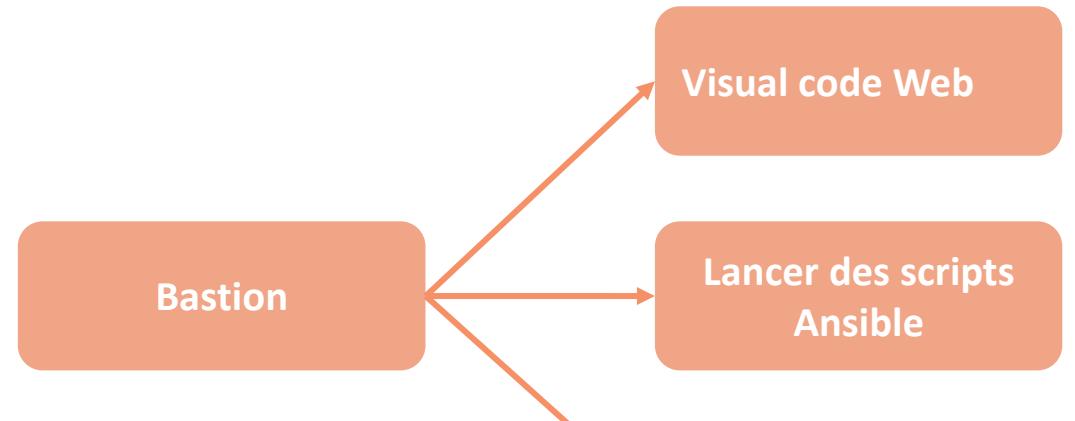
MISE EN PLACE ENVIRONNEMENT (2/3)



```
EXPLORER WORKSHOP-TP tp2 ! setup-petclinic-env.yml ... tp2 > ! setup-petclinic-env.yml
85 ignore_errors: yes # la creation de la base echoue si la base existe deja
86 ansible.builtin.command:
87   psql -p5433 -c "CREATE DATABASE petclinic OWNER petclinic"
88
89 # Define Postgresql as a Service
90 # TODO
91
92 ## Play application petclinic
93 # TODO
94
95 # Install OpenJdk 11
96 # TODO
97
98 # Create folder /app/
99 # TODO
100
101 # Install petclinic into /app/
102 # TODO
103
104
```

VMs en debian 11

Login : ansible
Password : ansible



/home/ansible/
petclinic/
binaries/
contient le binaire de l'applicatif java petclinic
src/
contient les sources de l'applicatif java petclinic
workspace/
reminders
contient des aides mémoires (commandes ssh,
emplacement des logs,...)
bonus
contient des scripts ansibles d'init (non utilisé pour les tps)
workspace-tp
contient les sources pour les TPs
workspace-tp-answers
contient les solutions des TPs



FIL ROUGE DU TP

MISE EN PLACE ENVIRONNEMENT (3/3)

Générer une clef
SSH

```
ansible@bastion-0:~$ ssh-keygen
```

Generating public/private rsa key pair.

Enter file in which to save the key (/home/ansible/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

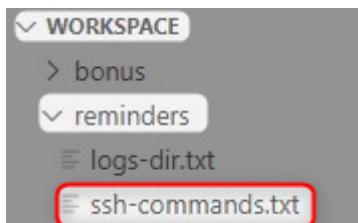
Enter same passphrase again:

Your identification has been saved in /home/ansible/.ssh/id_rsa

Your public key has been saved in /home/ansible/.ssh/id_rsa.pub

Copie de la clef SSH
sur les VMS

Commandes sous :



```
ansible@bastion-0:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub ansible@appdev01
```

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansible/.ssh/id_rsa.pub"

The authenticity of host 'appdev01 (10.0.35.78)' can't be established.

ECDSA key fingerprint is SHA256:6cj9EfFdqHVVBBFDzPBojbpqR+2STG7JsHU+j3eOkwE.

Are you sure you want to continue connecting (yes/no/[fingerprint])? Yes

ansible@appdev01's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'ansible@appdev01'"

and check to make sure that only the key(s) you wanted were added.

Liste VMs :

- DEV
 - appdev01
 - lbdev01
 - bdddev01
- REC
 - webrec01
 - webrec02
 - lbrec01
 - bddrec01



FIL ROUGE DU TP

MODE OPÉRATOIRE TP

- La formation est constituée de 8 TPs
- Chaque TP contiendra des squelettes de scripts à enrichir
- Chaque TP s'appuiera sur les réponses fournies du précédent TP (à l'exception du TP2)
 - Vous ne serez pas bloqué si vous n'avez pas terminé un TP
 - A la fin de chaque TP, la réponse sera présentée par les formateurs pour vous permettre de démarrer le TP suivant
 - Nous vous engageons à ne pas regarder les réponses du TP en cours. Expérimitez & sollicitez les formateurs en cas de questions / difficultés / blocages



Ansible s'apprend en expérimentant



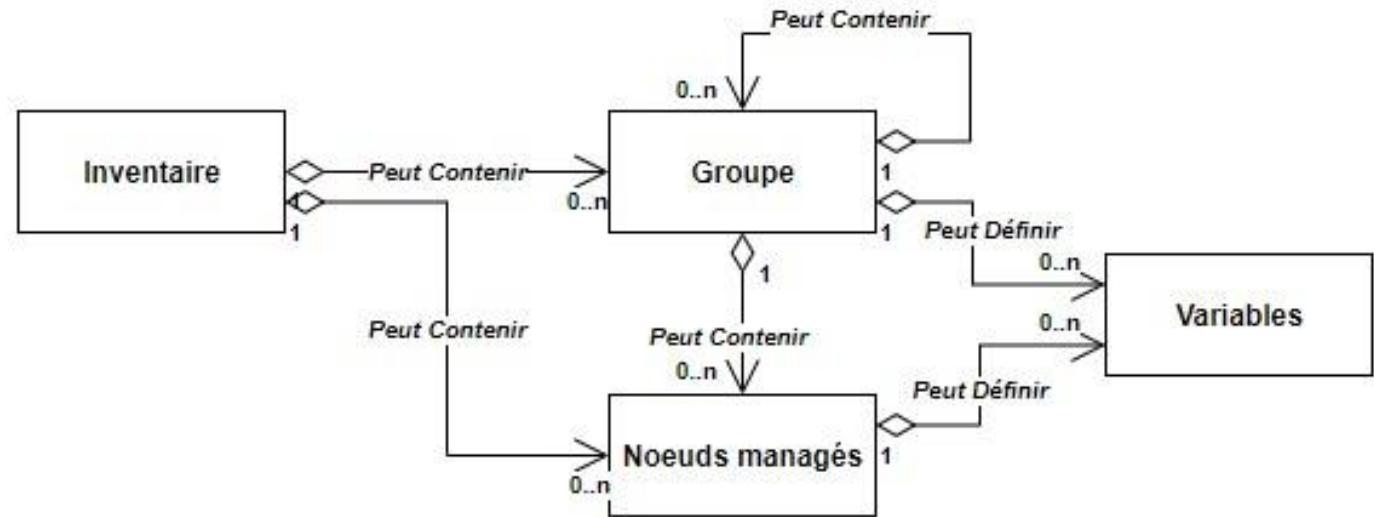
INVENTAIRE

BIG PICTURE



Inventaire Ansible

- Ensemble des serveurs managés par ansible
- Un inventaire peut définir
 - des groupes de serveurs managés
 - des serveurs managés sans groupe
- Un groupe ou un serveur peut définir des variables qui sont héritées pour les entités filles de l'inventaire
- NB.
 - Un groupe peut appartenir à un inventaire ou à un groupe de l'inventaire
 - Un nœud peut appartenir à plusieurs groupes d'un inventaire
 - Le groupe built-in '**all**' contient tous les nœuds managés de l'inventaire
 - Le groupe built-in '**ungrouped**' contient tous les nœuds managés n'appartenant à aucun groupe autre que '**all**'



INVENTAIRE

EXEMPLES

inv.exemple.ini

```
[load-balancers]
lb01
lb02

[web]
web01
web02
web03

[contracts-bdd-master]
bddcontractsM01

[contracts-bdd-slave]
bddcontractsS01
bddcontractsS02

[contracts-bdd:children]
contracts-bdd-master
contracts-bdd-slave

[contracts-bdd:vars]
custom_var=30
```

un groupe

un noeud

autre notation
web[01:03]

les sous groupes de
contracts-bdd

les variables de contracts-bdd

inv.exemple.yml

```
all:
  children:
    load-balancers:
      hosts:
        lb01:
        lb02:
    web:
      hosts:
        web01:
        web02:
        web03:
    contracts-bdd:
      children:
        contracts-bdd-master:
          hosts:
            bddcontractsM01:
        contracts-bdd-slave:
          hosts:
            bddcontractsS01:
            bddcontractsS02:
      vars:
        custom_var=30
```



Il est possible d'utiliser une **connexion** dite '**local**' lorsque ansible est installé sur la machine sur laquelle sera exécutée les commandes ansible



inventaire dynamique : hors scope de cette formation, voir 'pour aller plus loin'



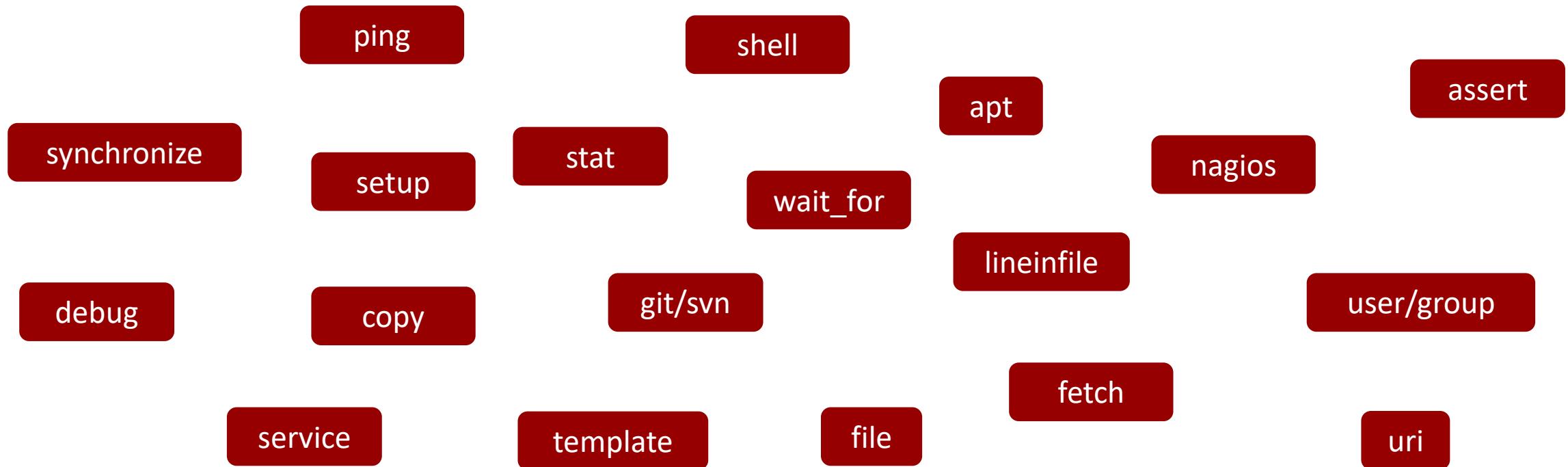
Pour aller plus loin : <https://docs.ansible.com/ansible/latest/plugins/inventory.html#inventory-plugins>



MODULES / COLLECTIONS

BRIQUE ELEMENTAIRE D'ANSIBLE

- Les modules (ansible 2.9) / collections (ansible 2.10+) définissent les traitements qu'Ansible sait opérer de base
- Ils contiennent notamment l'intelligence pour passer de l'état existant à l'état cible et respectent l'idempotence (*sauf mention contraire, ex shell / command*)



- Il est possible d'écrire ses propres modules / collections (en python)



MODULES DOCUMENTATION

- Le site internet
 - Ansible 2.9 :
 - https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html
 - Ansible latest :
 - https://docs.ansible.com/ansible/latest/collections/all_plugins.html
- La commande ‘**ansible-doc**’
 - Lister les modules : ansible-doc –list
 - Afficher la doc d'un module : ansible-doc ping
 - Afficher des exemples d'utilisation : ansible-doc -s ping
- Le code source <https://github.com/ansible/ansible.git>



MODULE

USAGE VIA UNE LIGNE DE COMMANDE

- Accès à l'aide en ligne



Code : aide en ligne

```
$ ansible -h
```

- Directement via une ligne de commande



Code : lancement ansible

```
$ ansible [all|nom_groupe|nom_noeud] -i [inventaire] -m [nom module] [args...]
```

-i pour indiquer l'inventaire.

Il peut s'agir du répertoire / fichier contenant l'inventaire ; ou des hosts / groupes séparés par une virgule



Pour activer les traces :

ajouter -v, -vv, ..., -vvvv pour avoir un niveau de trace de plus en plus élevé



-a : permet de préciser les arguments d'un module

Ex : -a "param1=123 param2=456"



Lorsque python n'est pas installé sur le noeud managé, vous pouvez utiliser le module raw. C'est l'un des rares modules qui n'a pas besoin de python.



Code : usage de raw pour installer python

```
$ ansible all -i inventory -m raw -a 'apt-get update && apt-get -y install python' --become
```



RÉCAPITULATIF

Qu'est-ce qu'ANSIBLE ?

Qu'apporte ANSIBLE ?

Comment communique ANSIBLE ?

Qu'est-ce que la programmation par état ?

Que sont les modules ANSIBLE ?

A quoi sert l'inventaire ANSIBLE ?



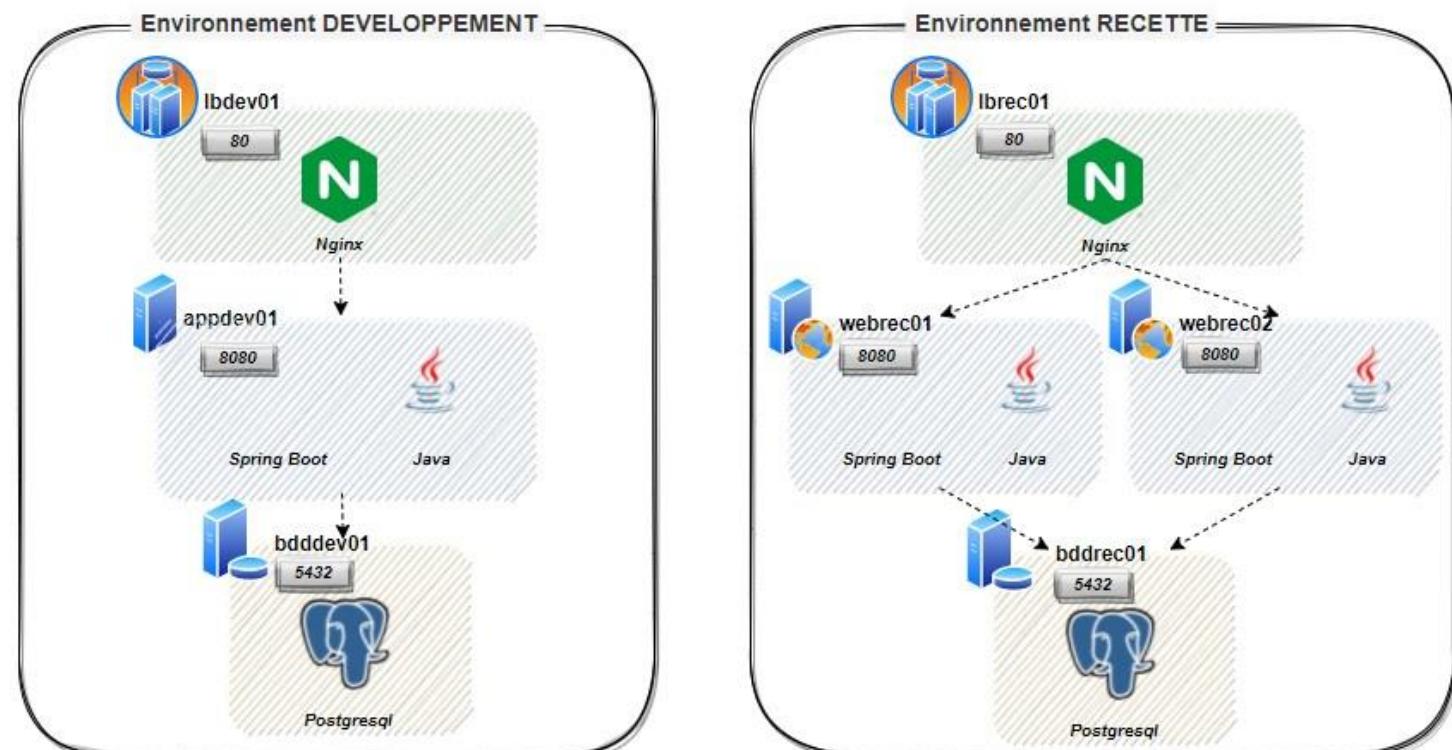
FIL ROUGE DU TP

USAGE DU MODULE PING



Objectif : Tester le module Ping

- Définir les inventaires du fil rouge (dev et recette)
 - Comment voulez-vous faire?



FIL ROUGE DU TP

USAGE DU MODULE PING



10 min

Objectif : Tester le module Ping

- Créer les inventaires du fil rouge (dev et recette)

Environnement de Développement

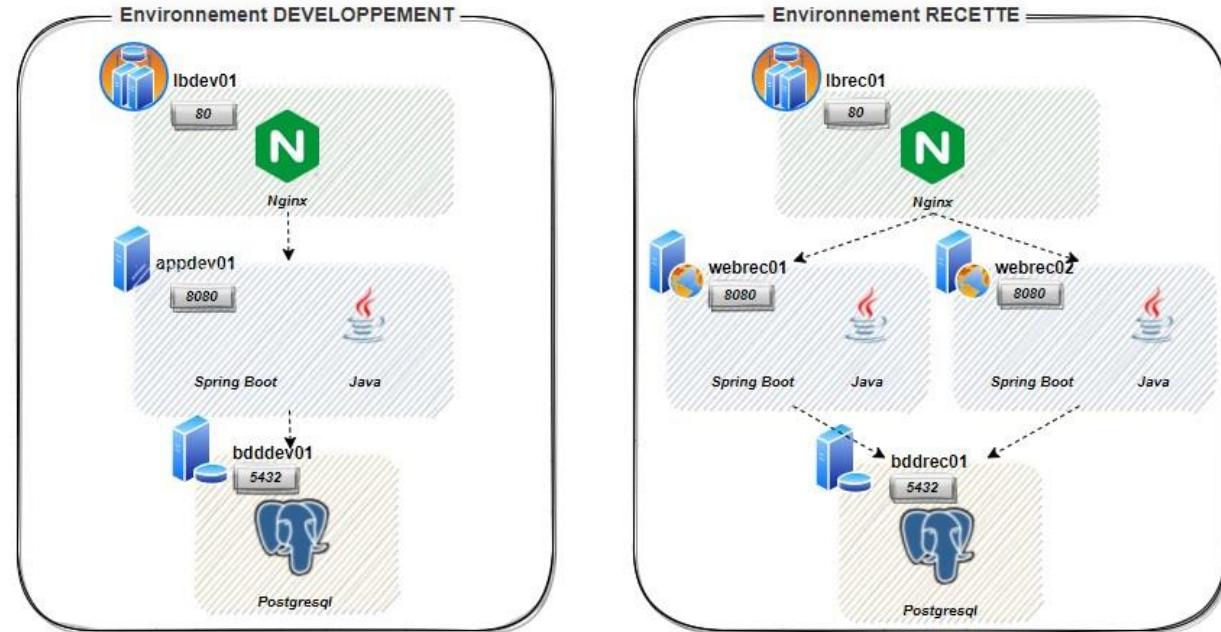
3 groupes :

- petclinic_loadbalancer_servers : 1 VM (lbdev01)
- petclinic_application_servers : 1 VM (appdev01)
- petclinic_database_servers : 1 VM (bdddev01)

Environnement de Recette

3 groupes :

- petclinic_loadbalancer_servers : 1 VM (ibreco1)
- petclinic_application_servers : 1 VM (webrec01 et webrec02)
- petclinic_database_servers : 1 VM (bddrec01)



Ou?

Dév. ➔ [Dossier TP]/workshop-tp/inventories/dev

Rec. ➔ [Dossier TP]/workshop-tp/inventories/rec



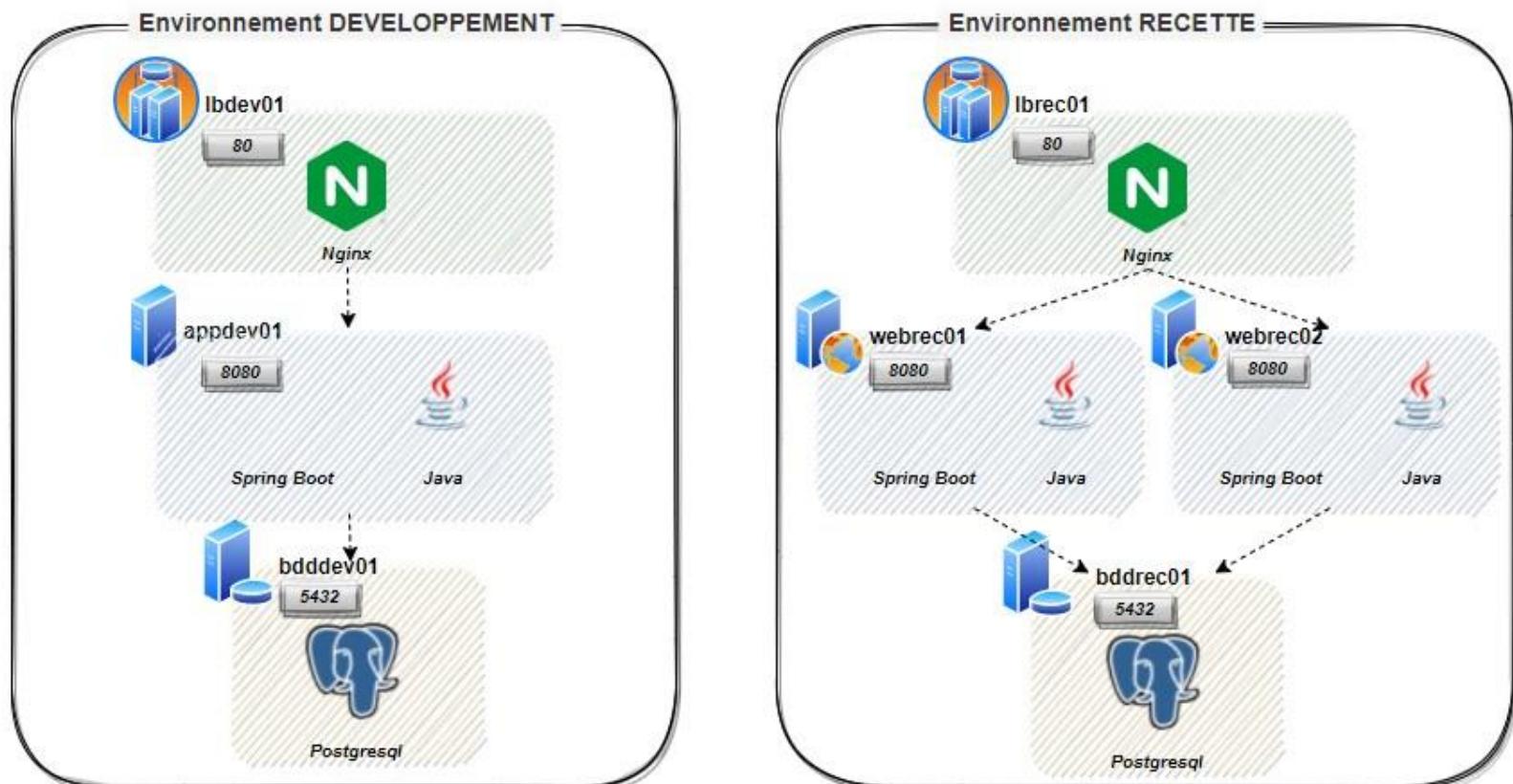
FIL ROUGE DU TP

USAGE DU MODULE PING



Objectif : Tester le module Ping

- Lancer le module ping sur l'environnement de développement et de recette



FIL ROUGE DU TP

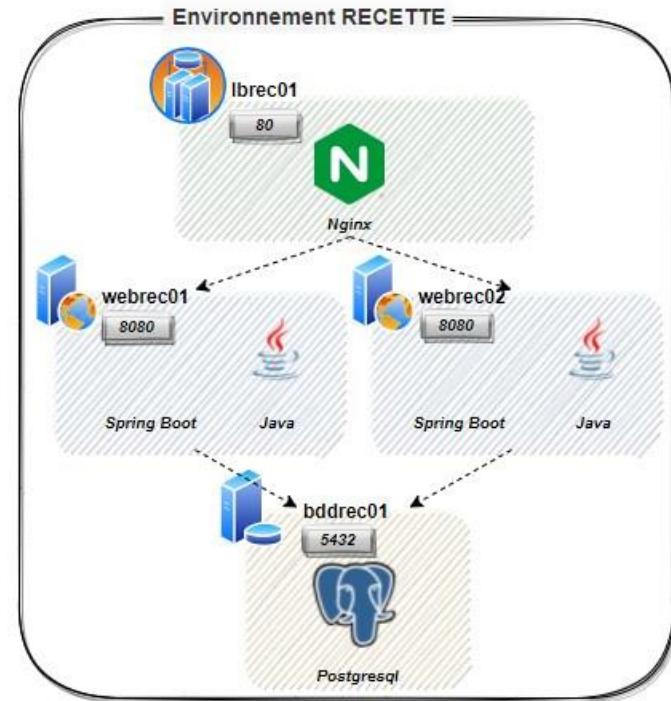
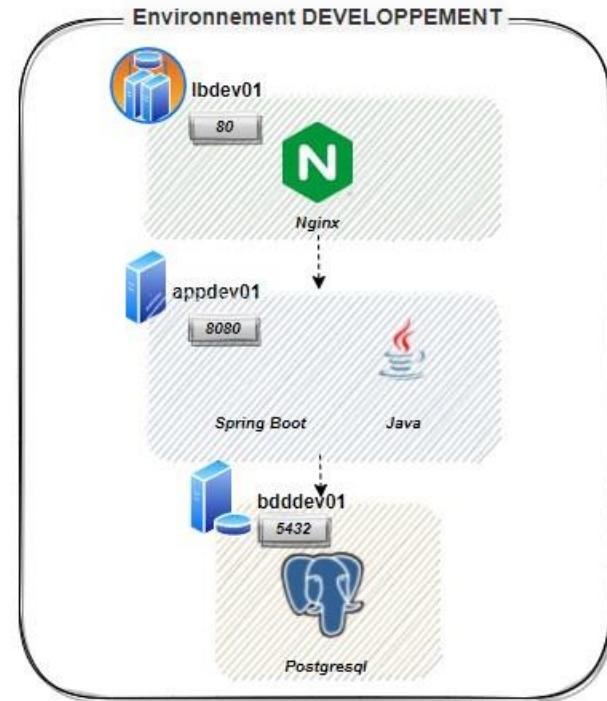
USAGE DU MODULE PING



5 min

Objectif : Tester le module Ping

- Lancer le module ping sur l'environnement de développement et de recette



Commandes :

```
ansible all -i [Dossier Ansible]/workshop-tp/inventories/dev -m ping  
ansible all -i [Dossier Ansible]/workshop-tp/inventories/rec -m ping
```



ANSIBLE TYPES

SCALAIRES, LISTES, DICTIONNAIRES & STRUCTURES (1/2)

- Ansible manipule 4 grands types : scalaires, listes /tableaux , dictionnaires, structures

- Les **types scalaires** (text, int, bool, ...)

- Les **listes ou tableaux**



Code : déclaration d'une variable version de type liste

```
vars:  
  version:  
    - v1  
    - v2
```

```
vars:  
  version: ["v1", "v2"]
```

En général plus visuel sous cette forme



Code : accès à une valeur d'une liste

```
current_version: "{{ version[0] }}"
```



Pour tester le type d'une variable : `ma_variable is string / number / float ... / mapping / iterable`



Pour aller plus loin : https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html



ANSIBLE TYPES

SCALAIRES, LISTES, DICTIONNAIRES & STRUCTURES (2/2)

■ Les dictionnaires



Code : déclaration d'une variable users de type dictionnaire

```
vars:  
  users:  
    - usr1: maria  
    - usr2: peter
```

```
vars:  
  users: { "usr1" : "maria", "usr2" : "peter"}
```



Code : accès à une valeur d'un dictionnaire

```
current_user: "{{ users['usr1'] }}"  
current_user: "{{ users.usr1 }}"
```

■ Les structures imbriquées (nested)



Code : déclaration d'une variable cidrs de type nested

```
vars:  
  cidrs:  
    production:  
      vpc: "172.31.0.0/16"  
    staging:  
      vpc: "10.0.0.0/24"
```



Code : accès à une valeur d'une structure

```
current_cidr: "{{ cidrs['production']['vpc'] }}"
```



ANSIBLE VARIABLES

VARIABLES BUILT-IN

- Ansible propose plusieurs variables réservées qui sont pré-renseignées par Ansible
- Ces variables ne peuvent être forcées par le script
- Parmi celles-ci :
 - **groups** : un dictionnaire de tous les groupes & serveurs de l'inventaire
 - **group_names** : la liste des groupes auxquels appartient le serveur managé en cours
 - **hostvars** : un dictionnaire avec l'ensemble des variables pour chaque serveur de l'inventaire
 - **inventory_hostname** : le nom du serveur managé en cours



Les variables précédentes permettent par exemple de récupérer les hosts d'un cluster composé de plusieurs éléments lors de son déploiement.



ANSIBLE FACTS

PLAY ET FAITS (1/2)

- Ansible peut récupérer des informations très détaillées sur les hosts appartenant à l'inventaire.
- Ces informations sont accessibles via le dictionnaire built-in **ansible_facts**
- Cette récupération est automatique en début de chaque script.
Elle peut être désactivée.



Code : exemple d'un script désactivant une collecte de faits

```
- hosts: groupe44
gather_facts: false
tasks:
- ...
```



Si les facts ne sont pas utilisés il vaut mieux désactiver.
Attention cependant car c'est très pratique ;)



ANSIBLE FACTS

PLAY ET FAITS (2/2)

- Une fois les faits sur un hôte collectés, on peut les utiliser



Code : exemple d'un script utilisant un fait

```
- hosts: all
  tasks:
    - name: compilation de l'application
      command: "make -j {{ ansible_processor_cores }}"
```

- La liste des facts disponibles peut être trouvé directement au sein de la documentation ansible

https://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html#ansible-facts

TIP

Ne pas hésiter à parcourir l'ensemble des facts disponibles pour en avoir une idée globale



ANSIBLE FACTS MODULE SETUP

- Le module **setup** permet de récupérer les faits

Code : *appel du module setup en filtrant certains attributs*

```
ansible all -i inventories/formation -m setup -a  
'filter=ansible_distribution,ansible_distribution_version,  
ansible_os_family,ansible_processor,ansible_python'
```

ansible_processor est une liste (cf. [])

ansible_python est un dictionnaire (cf. {})

```
"ansible_facts": {  
    "ansible_distribution": "Ubuntu",  
    "ansible_distribution_version": "20.04",  
    "ansible_os_family": "Debian",  
    "ansible_processor": [  
        "0",  
        "GenuineIntel",  
        "Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz",  
        "1",  
        "GenuineIntel",  
        "Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz"  
    ],  
    "ansible_python": {  
        "executable": "/usr/bin/python3",  
        "has_sslcontext": true,  
        "type": "cpython",  
        "version": {  
            "major": 3,  
            "micro": 10,  
            "minor": 8,  
            "releaselevel": "final",  
            "serial": 0  
        },  
        "version_info": [  
            3,  
            8,  
            10,  
            "final",  
            0  
        ]  
    },  
    "discovered_interpreter_python": "/usr/bin/python3"  
}
```

Pour aller plus loin :

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/setup_module.html

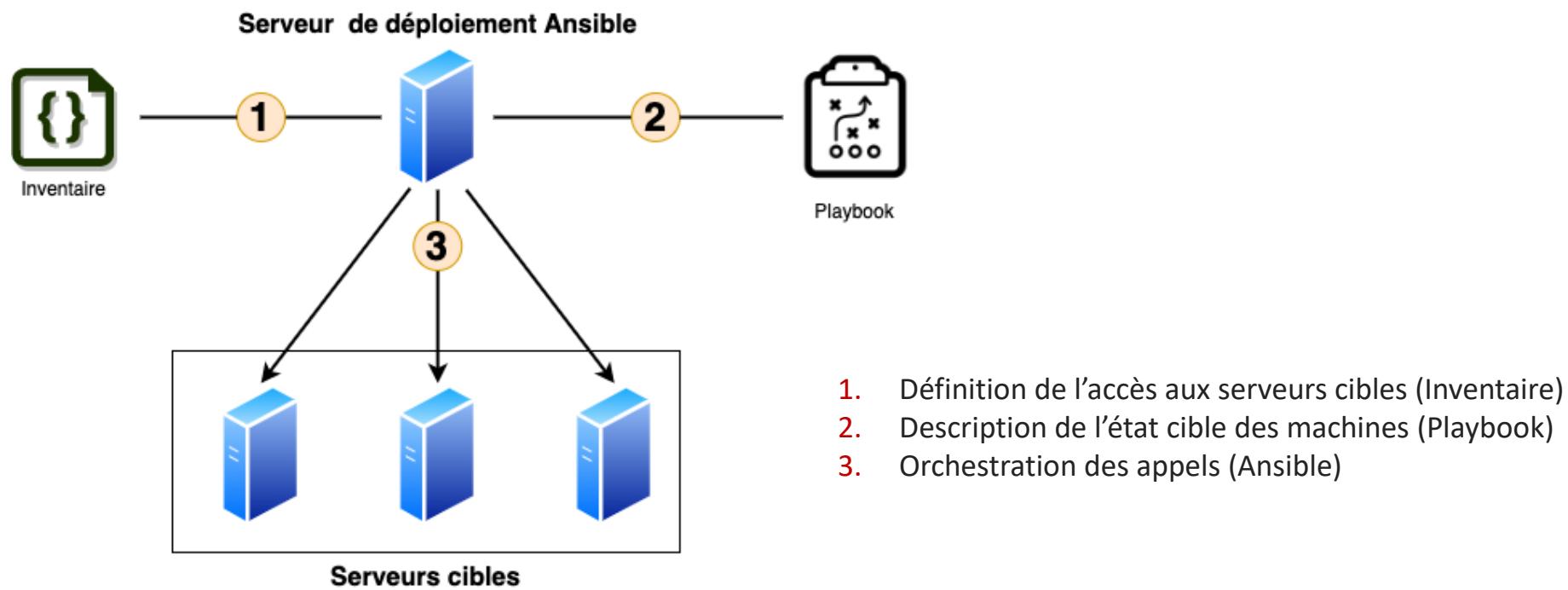


PLAYBOOK DÉFINITION (1/2)



■ Objectifs

- Décrire un **enchainement rejouable** d'appel à des modules dans un fichier
- Attribuer les commandes à jouer en fonction des machines et/ou groupes de l'inventaire
- **Eviter de taper les commandes à la main**

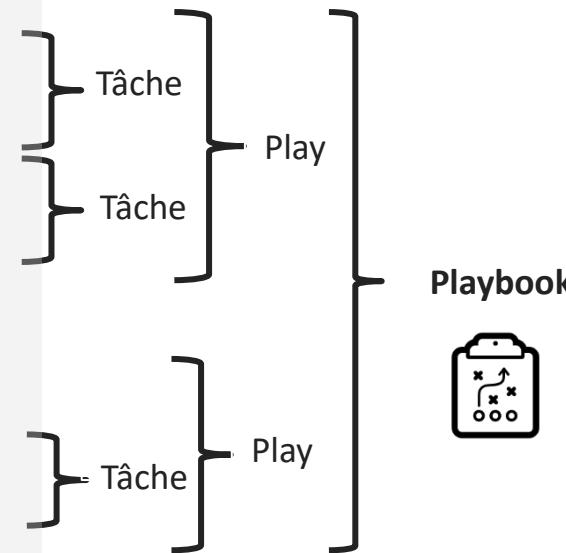


PLAYBOOK DÉFINITION (2/2)



Code : structure d'un playbook

```
---  
# Mon super play  
- hosts: all  
  tasks:  
    - name: fait un truc  
      module1:  
        argument: valeur44  
    - name: fait un autre truc  
      module2:  
        argument: valeur66  
  
# Un autre play  
- hosts: groupe2:groupe3  
  tasks:  
    - name: fait un machin  
      module3:  
        argument: valeur99
```



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: started  
        enabled: true
```



Le fichier Playbook est au format YAML



En général, 1 Playbook = 1 Play



PLAYBOOK EXÉCUTION (1/3)



- Ansible se connecte à l'hôte et récupère les faits
- Il traduit les tâches en commandes, qu'il transmet aux machines via SSH
- 4 résultats possibles :



Code : ensemble des états retour possible

changed
ok
skipped
failed

changed L'état constaté n'est pas l'état attendu, Ansible a exécuté avec succès la tâche pour atteindre l'état cible.

ok L'état constaté est l'état attendu. Aucune action n'a été réalisée. (**idempotence**)

skipped Une condition liée à l'exécution de la tâche n'était pas réalisée pour réaliser la tâche.

failed Une erreur est survenue durant l'exécution de la tâche.



PLAYBOOK EXÉCUTION (2/3)



- Une **tâche** doit être **intégralement exécutée sur toutes les machines** concernées avant de passer à la suivante
- Un **play** doit être **intégralement exécuté sur toutes les machines concernées** avant de passer au suivant
- Un **playbook** est **exécuté au moyen d'une stratégie** qui définit le flux d'exécution des tâches

notions
expert

Stratégie	Description
linear (default)	Le Playbook s'exécute de manière linéaire, chaque tâche est exécutée dans l'ordre et la tâche suivante n'est pas lancée tant que tous les hôtes n'ont pas terminé la tâche en cours.
debug	L'exécution se fait dans un mode de débogage qui permet à l'utilisateur d'exécuter interactivement le playbook à des fins de dépannage.
free	Le playbook est exécuté de manière linéaire, mais chaque hôte exécute les tâches à son propre rythme et n'a pas besoin que tous les hôtes aient terminé une tâche avant de passer à la suivante.
host_pinned	Fonctionnant comme la stratégie free, host_pinned exécutera les tâches aussi vite qu'il le peut sur le nombre d'hôtes spécifié dans le mot-clé serial (batch de machines), en démarrant immédiatement un hôte dès qu'un autre se termine.



Code : forcer la stratégie à « free »

```
- hosts: groupe44
strategy: free
tasks:
  - ...
```



Code : Modifier le fichier de configuration d'Ansible

```
[defaults]
strategy = free. # forcer la stratégie à « free »
```



PLAYBOOK EXÉCUTION (3/3)



notions expert

- Un play est par défaut joué en parallèle sur un ensemble de machines (par défaut 5) via l'option **fork**.



Code : forcer le // à 10

```
$ ansible-playbook -f 10 my_playbook.yml
```



Code : Modifier le fichier de configuration d'Ansible

```
[defaults]  
forks = 10. # forcer le // à 10
```

- Il peut aussi être joué par **lotissement**, utile notamment lors de rolling update



Code : exemple d'un play par lot de 10 serveurs

```
- hosts: groupe44  
serial: 10  
max_fail_percentage: 25%  
tasks:  
- ...
```



N lots comprenant au max.
10 serveurs du groupe44



Code : exemple d'un play par lot de 20% d'un groupe de serveurs

```
- hosts: groupe44  
serial: 20%  
tasks:  
- ...
```



5 lots comprenant 20% des
serveurs du groupe44



Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html



PLAYBOOK

APPELER UN PLAYBOOK



- Appeler un playbook sans paramètre



Code : appel d'un playbook portant sur un inventaire

```
$ ansible-playbook -i inventories/dev playbook.yml
```

- Appeler plusieurs playbooks



Code : appel de plusieurs playbooks

```
$ ansible-playbook -i inventories/dev playbook1.yml playbook2.yml
```



PLAYBOOK

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      package:
        name: nginx
    - name: template conf file
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
    - name: enable / start Nginx
      service:
        name: nginx
        state: started
        enabled: true
```



PLAYBOOK

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: started  
        enabled: true
```

1^{ère}
exécution

changed

changed

changed



PLAYBOOK

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: started  
        enabled: true
```

1^{ère}
exécution

changed

2^e
exécution

ok

changed

ok

changed

ok

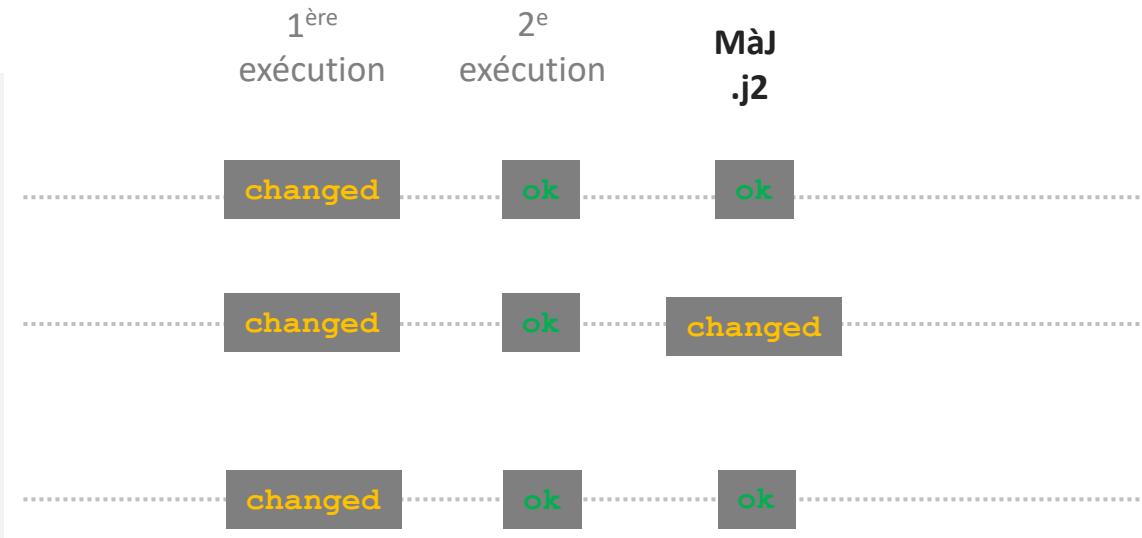
PLAYBOOK

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: started  
        enabled: true
```



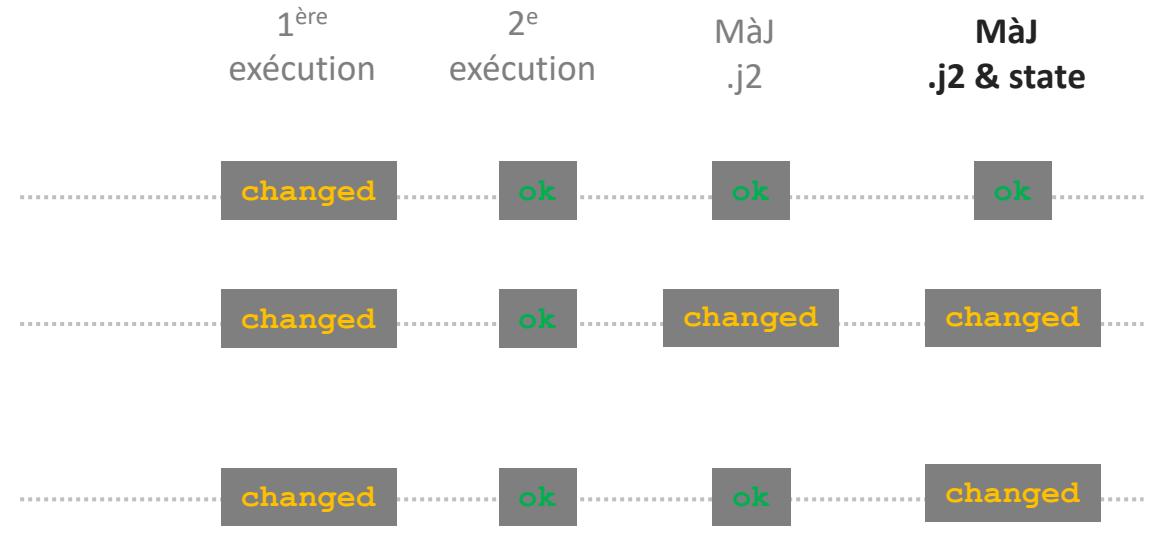
PLAYBOOK

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: restarted  
        enabled: true
```

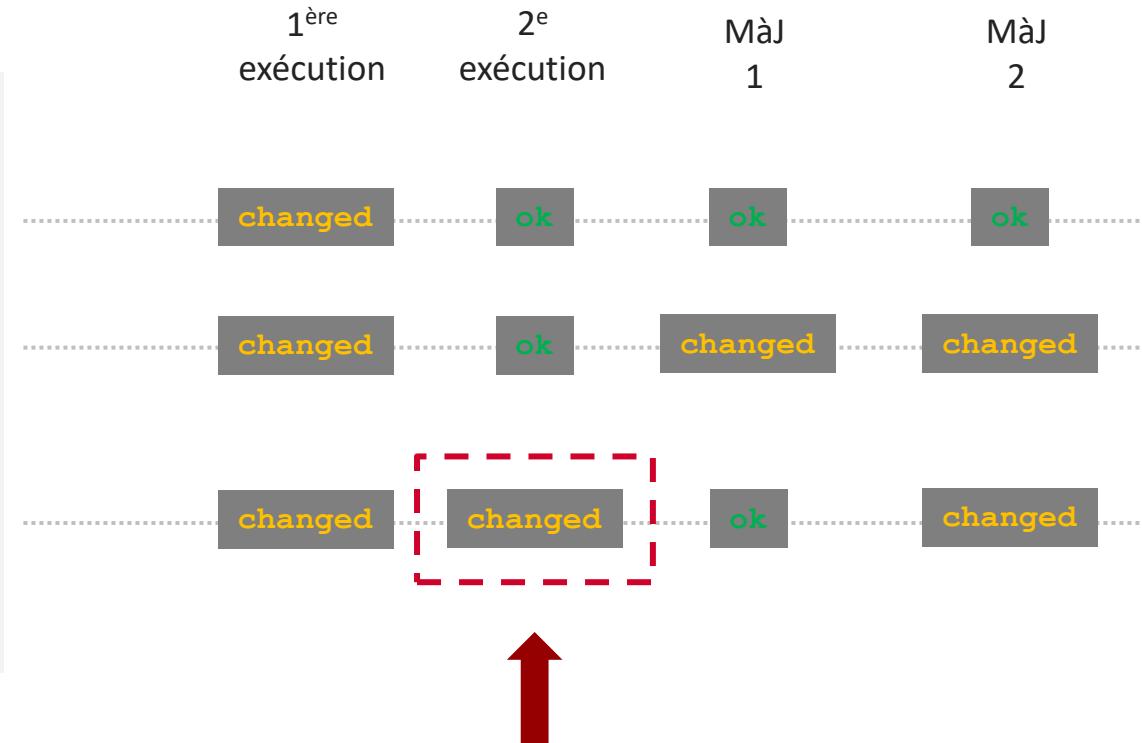


PLAYBOOK 1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx

```
---  
- hosts: webservers  
  tasks:  
    - name: Install Nginx  
      package:  
        name: nginx  
    - name: template conf file  
      template:  
        src: nginx.conf.j2  
        dest: /etc/nginx/nginx.conf  
    - name: enable / start Nginx  
      service:  
        name: nginx  
        state: restarted  
        enabled: true
```



Si on relance l'exécution 2 avec la modification précédente

DIVERS PLAY ET UTILISATEURS

- Un play peut être lancé avec sudo via l'option « **become** »



Code : exemple d'un play lancé en sudo

```
- hosts: groupe44
become: true
tasks:
- ...
```



A noter : become permet de lancer les tâches en tant que root via la commande sudo. Les utilisateurs autorisés à utiliser sudo sont configurés par les administrateurs.

- « **become_user** » indique sous quel utilisateur le play sera lancé



Code : exemple d'un play lancé en tant qu'un utilisateur toto

```
- hosts: groupe44
become: true
become_user: toto
tasks:
- ...
```



*A noter : become_user permet de préciser l'utilisateur bénéficiant de l'élevation de privilège.
Equivalent de la commande « su » sous linux*



DIVERS

PLAY ET REGISTER

- Il peut être utile de capturer la sortie des modules d'un play exécuté par Ansible via l'instruction **register**.



Code : exemple d'un module exploitant register

```
- hosts: all
  tasks:
    - name: print hello world
      command: echo "hello there"
      register: echo_return
    - debug: msg="{{ echo_return.stdout }}"
    - debug: msg="{{ echo_return.stderr }}
```

Permet d'afficher les sorties standards

- Différentes propriétés sont accessibles au sein de la variable déclarée via register :

```
"echo_return": {
  "failed": false,
  "changed": true,
  "cmd": [...],
  "rc": 0,
  "start": "...",
  "end": "...",
  "delta": "...",
  "stderr": "...",
  "stderr_lines": [...],
  "stdout": "...",
  "stdout_lines": [...],
  "warnings": []
}
```

Le module est-il en échec ou a-t-il changé l'état du host ?

La commande exécutée et son code retour

L'horaire de début, de fin et le temps écoulé pour exécuter le module

Les sorties standards et d'erreur avec les éventuels warnings

TIP std*_lines correspond à std*.split('\n')

Pour aller plus loin : https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#registering-variables



LINTING (1/2)

ANSIBLE-LINT

- Ansible propose un utilitaire « **ansible-lint** » pour s'assurer du respect de certaines règles de développement built-in



Code : exemple d'usage d'ansible-lint

```
> ansible-lint -p workshop-tp/tp2
```



A noter : si l'argument de **-p** est un répertoire, **ansible-lint** vérifie l'ensemble des fichiers présents dans le répertoire (et sous répertoire)

- Ces règles sont regroupées par catégorie



Code : exemple de résultat de ansible-lint

```
WARNING Listing 10 violation(s) that are fatal
workshop-tp/tp2/myplaybook.yml:16: yaml[trailing-spaces]: Trailing spaces
workshop-tp/tp2/myplaybook.yml:17: yaml[trailing-spaces]: Trailing spaces
workshop-tp/tp2/myplaybook.yml:24: yaml[trailing-spaces]: Trailing spaces
workshop-tp/tp2/myplaybook.yml:26: name[play]: All plays should be named.
workshop-tp/tp2/myplaybook.yml:26: yaml[colons]: Too many spaces after colon
workshop-tp/tp2/myplaybook.yml:30: yaml[trailing-spaces]: Trailing spaces
workshop-tp/tp2/myplaybook.yml:33: fqcn[action]: Use FQCN for module actions, such `community.postgresql.postgresql_user`.
workshop-tp/tp2/myplaybook.yml:37: yaml[truthy]: Truthy value should be one of
workshop-tp/tp2/myplaybook.yml:44: fqcn[action]: Use FQCN for module actions, such `community.postgresql.postgresql_db`.
workshop-tp/tp2/myplaybook.yml:62: yaml[trailing-spaces]: Trailing spaces
Read documentation for instructions on how to ignore specific rule violations.
```

Rule Violation Summary		
count tag	profile	rule associated tags
1	name[play]	basic idiom
1	yaml[colons]	basic formatting, yaml
5	yaml[trailing-spaces]	basic formatting, yaml
1	yaml[truthy]	basic formatting, yaml
2	fqcn[action]	production formatting

```
Failed after min profile: 10 failure(s), 0 warning(s) on 1 files.
```

nom de la catégorie [nom de la règle]



A noter : il est possible de lancer les règles d'un seul profil avec l'option **--profile <nom du profil>**

Profil : min < basic < moderate < safety < shared < production

Un profil de niveau supérieur embarque les règles du niveau inférieur



Pour aller plus loin : <https://ansible.readthedocs.io/projects/lint/>



LINTING (2/2)

ANSIBLE-LINT

- L'argument ansible-lint qui va vous sauver : **--write**



Code : exemple d'usage d'ansible-lint

```
> ansible-lint -p <myplaybook> --write
```



--write va corriger automatiquement certaines erreurs de votre script ansible. Il va aussi reformater le script pour être consistant.



Code : ansible-lint sans write

Rule Violation Summary			
count	tag	profile	rule associated tags
4	name[play]	basic	idiom
62	yaml[cols]	basic	formatting, yaml
2	yaml[indentation]	basic	formatting, yaml
2	yaml[trailing-spaces]	basic	formatting, yaml
20	yaml[truthy]	basic	formatting, yaml
3	name[casing]	moderate	idiom
5	risky-file-permissions	safety	unpredictability
1	ignore-errors	shared	unpredictability
3	no-changed-when	shared	command-shell, idempotency
2	fqcn[action-core]	production	formatting
2	fqcn[action]	production	formatting



Code : ansible-lint avec write

Rule Violation Summary			
count	tag	profile	rule associated tags
4	name[play]	basic	idiom
3	name[casing]	moderate	idiom
5	risky-file-permissions	safety	unpredictability
1	ignore-errors	shared	unpredictability
3	no-changed-when	shared	command-shell, idempotency
2	fqcn[action-core]	production	formatting
2	fqcn[action]	production	formatting



RÉCAPITULATIF

Qu'est ce qu'un playbook ?

Qu'est-ce qu'une tâche ?

Quels sont les états finaux possible d'une tâche ?

Qu'est-ce que la programmation par état ?

Comment lance-t-on l'exécution d'un playbook ?



FIL ROUGE DU TP

UTILISATION DES PLAYBOOKS



2h

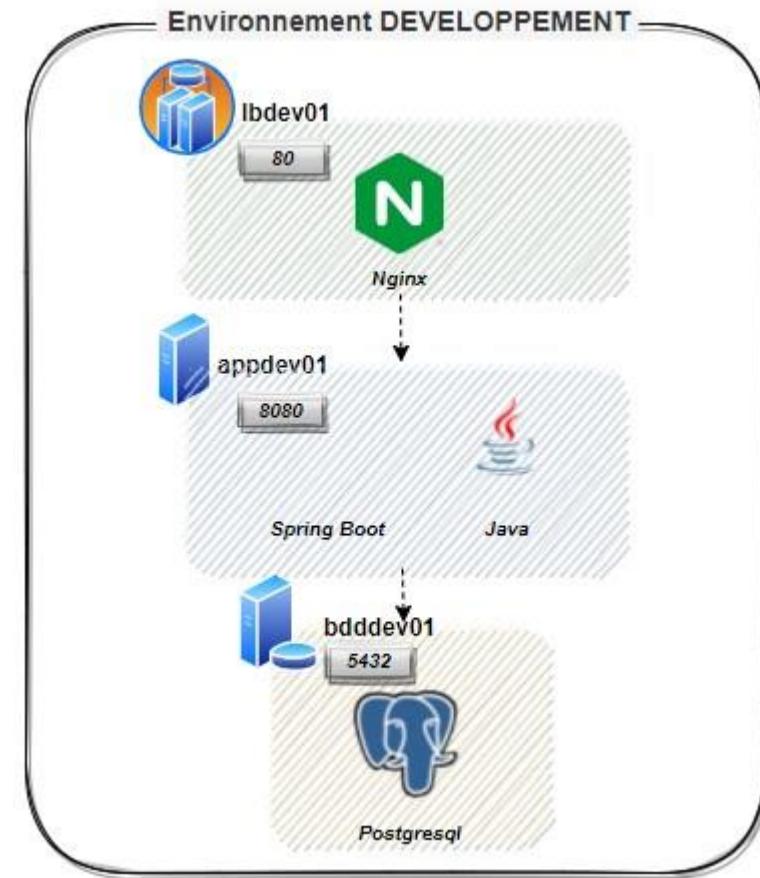
Objectif : Installer la plateforme de développement

- Ecrire le playbook
 - Installation NGINX (LoadBalancer)
 - Installation applicatif PetClinic

NB : les tâches relatives à la BDD sont déjà renseignées

TODO :

Compléter le playbook workshop-tp/tp2/setup-petclinic-env.yml
Lancer ansible-lint sur le playbook et corriger qq erreurs



A noter : du fait de la façon dont ansible-lint a été installé (via python pip3 au sein d'un environnement virtuel), il est nécessaire d'activer cet environnement virtuel via la commande activate / deactivate
> source ~/.venv/molecule/bin/activate
> ansible-lint
> deactivate



FIL ROUGE DU TP

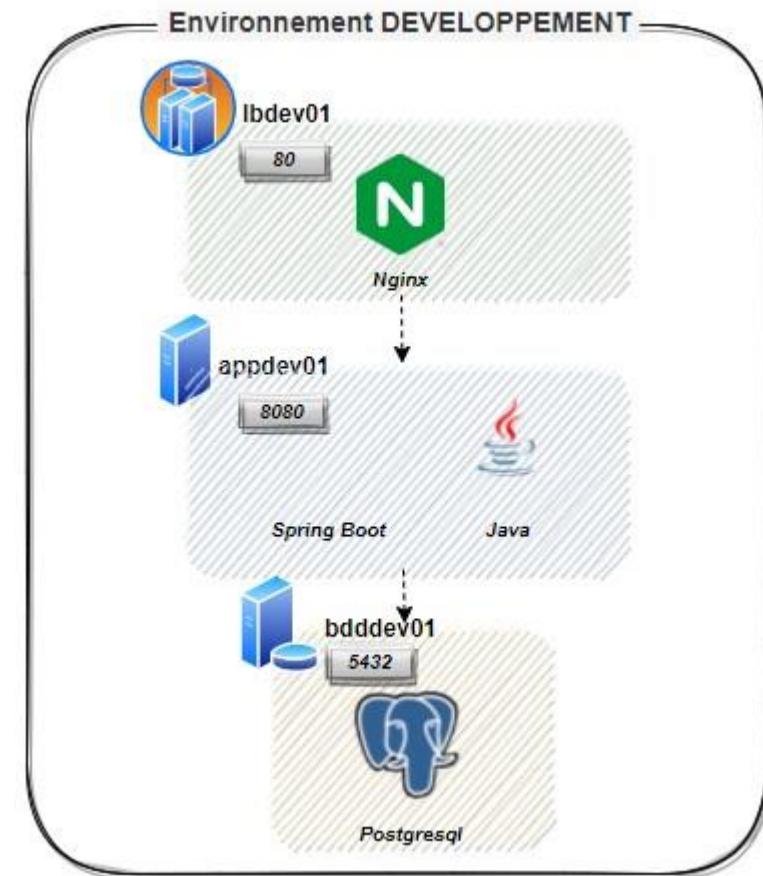
UTILISATION DES PLAYBOOKS



5 min

Objectif : Installer la plateforme de développement

- Utilisation du playbook sur la plateforme de développement
 - Quelles commandes pour lancer le playbook ?



FIL ROUGE DU TP

UTILISATION DES PLAYBOOKS



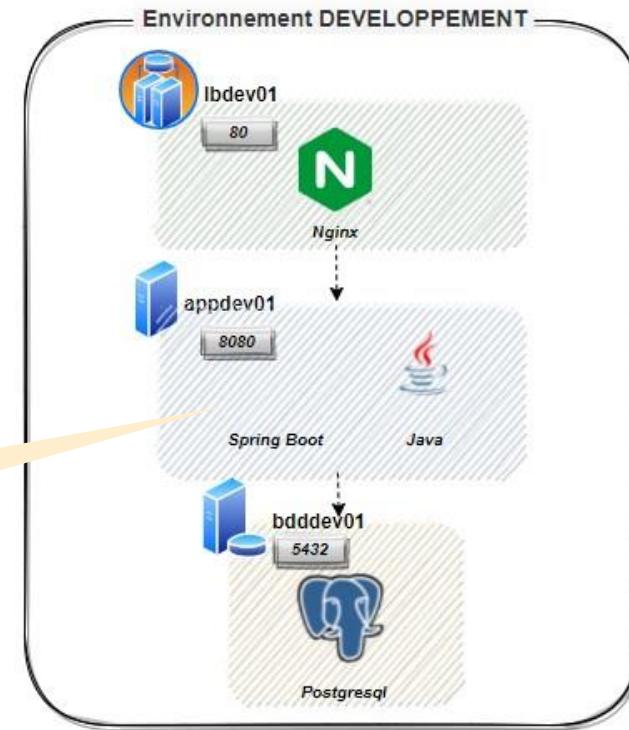
10 min

Objectif : Installer la plateforme de développement

- Utilisation du playbook sur la plateforme de développement



Pour démarrer l'application petclinic,
- se connecter en ssh sur appdev01
- cd /apps/petclinic
- . scripts/start-petclinic-postgresql-db.sh



Commandes (se positionner dans le répertoire racine du TP) :

```
ansible-playbook -i workshop-tp/inventories/dev workshop-tp/tp2/setup-petclinic-env.yml
```



FIL ROUGE DU TP

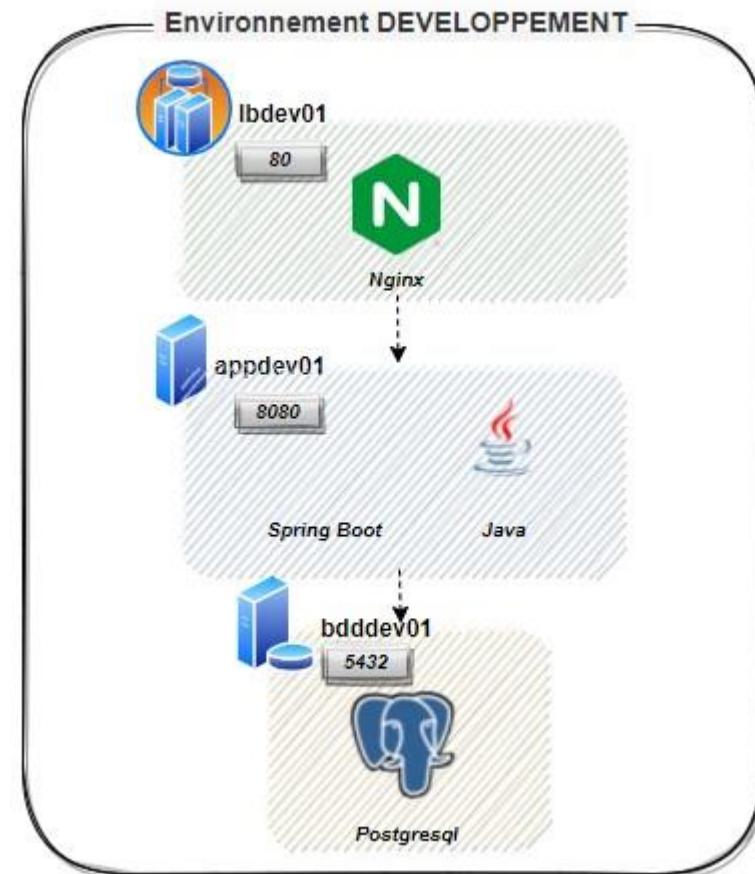
UTILISATION DES PLAYBOOKS



15 min

Objectif : Installer la plateforme de développement

- Idempotence
 - On relance le playbook sur la dev.
Quels sont les états?
 - On change le port du Nginx sur la machine.
On relance le playbook sur la dev.
Quels sont les états?



NOTIFY ET HANDLERS

DÉFINITION

- Les handlers sont des tâches spéciales
- Ces tâches sont déclenchées / exécutées que si une autre tâche a provoqué un changement et que le handler est notifié par la tâche qui a provoqué le changement



- **Un handler a une portée globale au niveau d'Ansible. Attention aux collisions sur le nommage des handlers.**
- **Un handler ne sera exécuté qu'une seule fois, même si plusieurs notifications ont eu lieu. Cette exécution aura lieu une fois toutes les tâches terminées.**



Exemple classique des handlers : redémarrer à chaud un service après un changement de configuration



Le mécanisme sous-jacent aux handlers est la notion d'évènement.
Ce mécanisme est à utiliser avec discernement pour éviter d'avoir des évènements dans tous les sens, rendant le playbook difficilement maintenable. #Platdespaghettis



NOTIFY ET HANDLERS

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx avec handler

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      ...
    - name: template conf file
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
    notify:
      - reload nginx
      - another event
  handlers:
    - name: ensures nginx restart
      listen : reload nginx
      service:
        name: nginx
        state: restarted
```



A noter : **reload nginx** est le nom de l'évènement qui sera notifié.
Il est possible de lancer plusieurs évènements au sein d'un notify.



A noter : le nom est **sensible à la casse** et peut contenir des espaces.

NOTIFY ET HANDLERS

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx avec handler

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      ...
    - name: template conf file
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
      notify:
        - reload nginx
  handlers:
    - name: ensures nginx restart
      listen : reload nginx
      service:
        name: nginx
        state: restarted
```

1^{ère}
exécution

changed
changed

change au niveau de la tâche déclarant le notify
=> évènement déclenchant le handler

changed



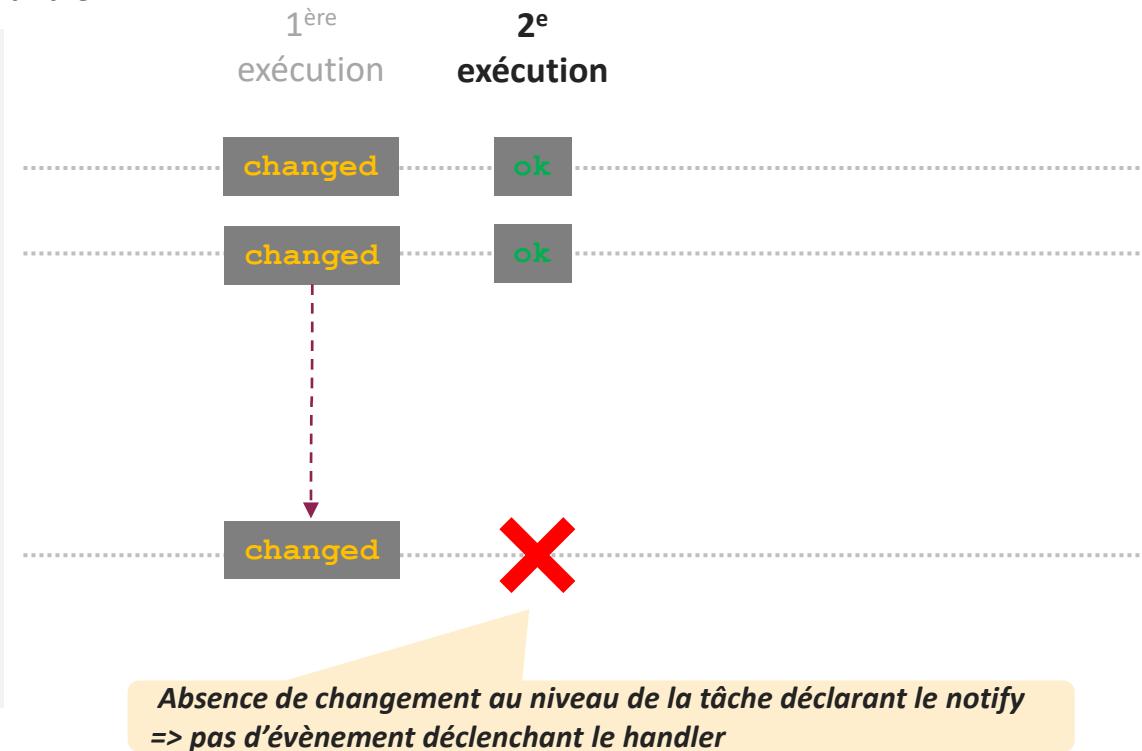
NOTIFY ET HANDLERS

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx avec handler

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      ...
    - name: template conf file
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
      notify:
        - reload nginx
    handlers:
      - name: ensures nginx restart
        listen : reload nginx
        service:
          name: nginx
          state: restarted
```



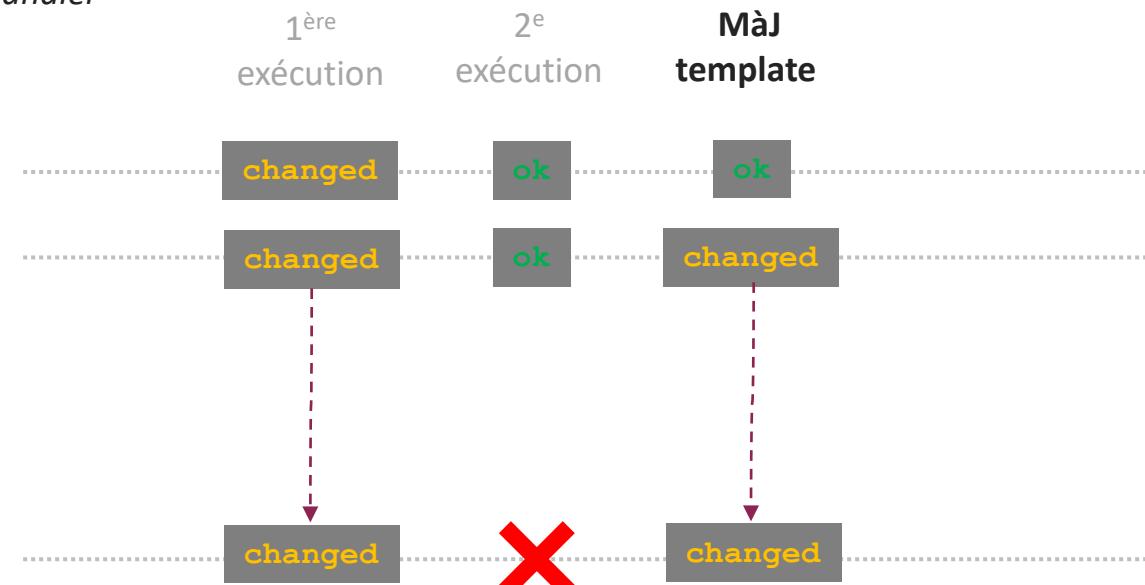
NOTIFY ET HANDLERS

1ER DÉPLOIEMENT VS MISE À JOUR



Code : exemple d'installation d'un serveur web Nginx avec handler

```
---
- hosts: webservers
  tasks:
    - name: Install Nginx
      ...
    - name: template conf file
      template:
        src: nginx.conf.j2
        dest: /etc/nginx/nginx.conf
      notify:
        - reload nginx
    handlers:
      - name: ensures nginx restart
        listen : reload nginx
        service:
          name: nginx
          state: restarted
```



NOTIFY ET HANDLERS

ORDRE D'EXÉCUTION DES HANDLERS

- Par défaut, les handlers sont exécutés à la fin du play
- On peut forcer l'exécution des handlers en attente avant la fin de toutes les tâches



Code : exemple d'un play forçant les handlers

```
- name: force handlers
  meta: flush_handlers
```

notions
avancées

- Les handlers sont exécutés dans l'ordre de leur déclaration dans les plays (cf code, 1 puis 2)



Code : exemple d'un play avec plusieurs handlers

```
- name: Verify apache installation
  hosts: webservers
  tasks:
    - name: Write the apache config file
      ...
      notify:
        - Restart apache
    - name: Write the tomcat config file
      ...
      notify:
        - Restart tomcat
  handlers:
    - name: Restart apache
      service:
        name: httpd
        state: restarted
    - name: Restart tomcat
      service:
        name: tomcat
        state: restarted
```

1

2



FIL ROUGE DU TP

TEST DES NOTIFY ET HANDLER



45 min

Objectif : Ne pas avoir à redémarrer l'instances nginx s'il n'y a aucune modification de configuration

- Modifier le fichier pet-clinic-env.yml du dossier tp3 (correction du TP précédent)
 - Utilisation des notify et handler!

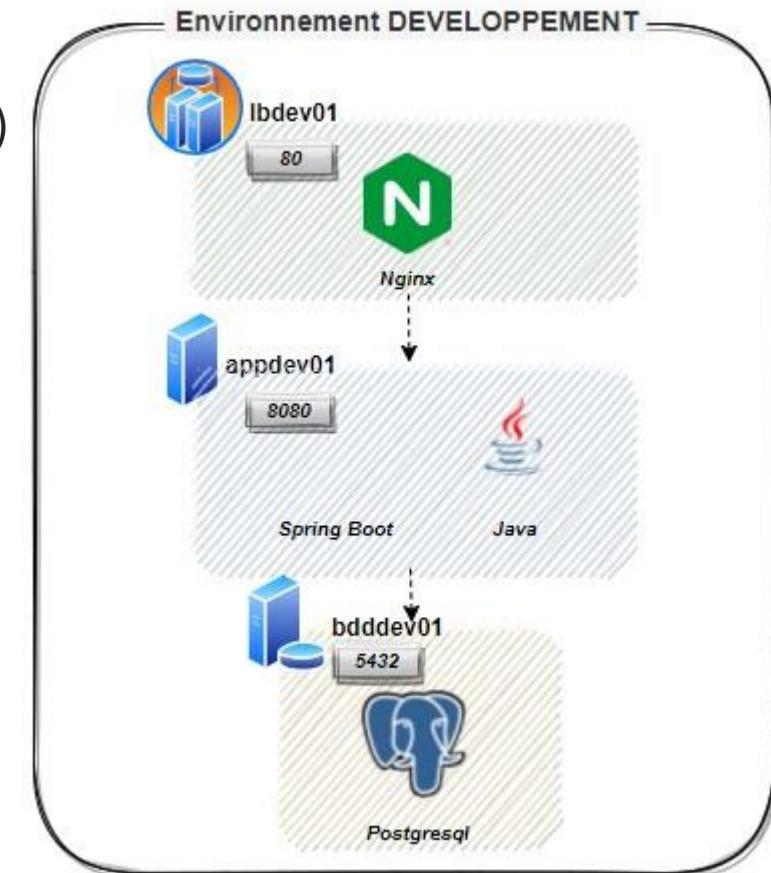
TODO :

Lancer le playbook workshop-tp/init-tp3.yml

Compléter le playbook workshop-tp/tp3/setup-petclinic-env.yml

TIP

Utiliser **ansible-playbook ... --limit Ibdev01** pour limiter l'exécution du playbook à Ibdev01 et non à la totalité de l'inventaire



FIL ROUGE DU TP

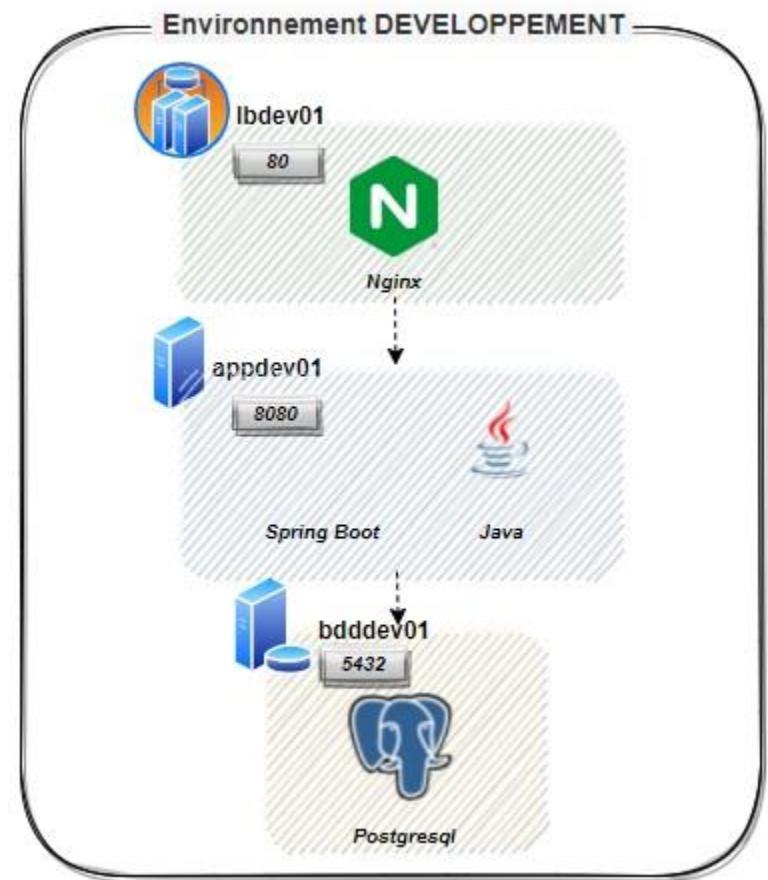
TEST DES NOTIFY ET HANDLER



10 min

Objectif : Ne pas avoir à redémarrer l'instance nginx s'il n'y a aucune modification de configuration

- Idempotence
 - On change le port d'exposition dans le fichier de configuration NGINX.
 - On relance le playbook.
 - Que se passe-t-il?



CONDITIONNEMENT

CONDITIONNER L' EXÉCUTION D'UNE TÂCHE (1/4)

- A partir de l'évaluation d'une expression libre
 - Il faut répéter la condition pour toutes les tâches



Code : exemple d'un play contenant un conditionnement

```
---  
- name: do some stuff  
  command: /bin/foo  
  register: cmd_result  
  
- name: do some more stuff  
  command: /usr/bin/do_stuff  
  when: "'bar' in cmd_result.stdout"
```

On utilise ici le résultat d'un register pour conditionner une autre tâche

CONDITIONNEMENT

CONDITIONNER L'EXÉCUTION D'UNE TÂCHE (2/4)

- A partir d'un changement



Code : exemple d'un play contenant un conditionnement

```
---  
- name: do some stuff  
  command: /bin/foo  
  register: cmd_result  
  
- name: do some more stuff  
  command: /usr/bin/do_stuff  
  when: cmd_result.changed
```

Le register permet de récupérer l'état de fin d'une tâche

CONDITIONNEMENT

CONDITIONNER L'EXÉCUTION D'UNE TÂCHE (3/4)

- Définir un comportement par défaut



Code : exemple d'un play contenant un conditionnement

```
$ ansible-playbook -i inv.ini install.yml -e really_do_it=1  
$ ansible-playbook -i inv.ini install.yml -e really_do_it=yes  
$ ansible-playbook -i inv.ini install.yml -e really_do_it=true
```



Les variables seront vues plus tard



Code : exemple d'un play contenant un conditionnement

```
---  
- name: do some more stuff  
  command: /usr/bin/do_stuff  
  when: "really_do_it | default(false) | bool"
```

CONDITIONNEMENT

CONDITIONNER L'EXÉCUTION D'UNE TÂCHE (4/4)

- A partir d'un fact



Code : exemple d'un play contenant un conditionnement

```
---
```

```
- name: do some more stuff
  command: /sbin/shutdown -t now
  when: ansible_os_family == "Debian"
```



Utile pour conditionner des déploiements en fonction du type de VM sur laquelle le déploiement s'effectue.

CONDITIONNEMENT

CONDITIONNER L'ÉTAT CHANGED OU FAILED

notions
avancées

- L'état changed ou failed d'un module / tâche peut être piloté par **changed_when** et **failed_when**



Code : exemple pour conditionner `changed_when`

```
---  
- name: do some stuff  
  command: /bin/foo  
  register: result  
  changed_when: "result.rc == 0 and 'OK' in result.stdout"
```



Utile pour gérer l'idempotence de module ne la supportant pas nativement (shell, command, ...)



Code : exemple pour conditionner `failed_when`

```
---  
- name: S'assurer que /tmp a bien à minima 1Gb  
  shell: "df -h /tmp|grep -v Filesystem|awk '{print $4}'|cut -d G -f1"  
  register: tmpspace  
  failed_when: "tmpspace.stdout | float < 1"
```

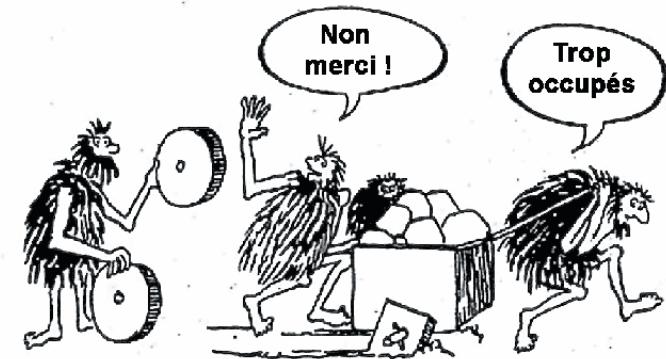


Utile pour s'assurer des prérequis d'une installation

RÔLES

GENERALITES (1/2)

- Les rôles ansible permettent de **réutiliser** du code ansible afin de développer plus rapidement des playbooks.
- A titre d'exemple, qq rôles possibles
- installer un nginx,
 - installer une base de données postgresql,
 - ...
- Chaque rôle doit être **générique**.
 - La **spécialisation** du rôle pour un projet s'appuiera sur
 - les **variables** du rôle qui seront passées en paramètres à l'appel du rôle
 - des **templates de fichiers** qui seront valorisés à l'exécution du rôle



RÔLES

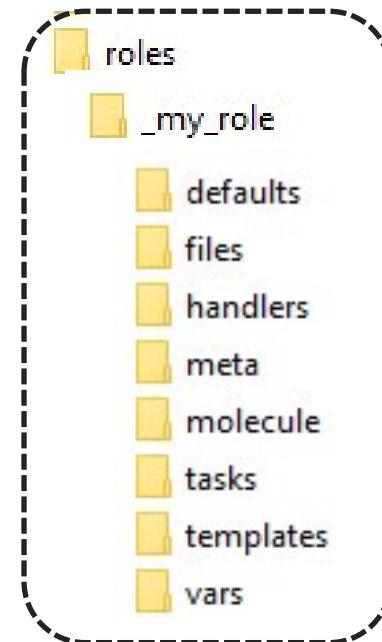
GENERALITES (2/2)

- Chaque rôle ansible s'appuie sur **une structure normalisée de répertoires** au niveau de votre projet.
 - **Tous les rôles d'un projet** sont contenus au sein du **répertoire roles** du projet
 - Chacun des sous répertoires du rôle doivent contenir **un fichier main.yml** (à l'exclusion des sous répertoires templates & files).
 - **Seul le sous répertoire tasks est obligatoire.**
- Pour **initialiser la structure** de répertoire d'un rôle, utiliser **ansible-galaxy**



Code : initialisation du role _my_role avec ansible-galaxy

```
$ cd roles  
$ ansible-galaxy init _my_role
```



RÔLES TASKS (1/2)

- Le fichier **tasks/main.yml** contient les différentes tâches qui seront lancées au moment de l'appel du rôle dans le playbook



Code : exemple de fichier tasks/main.yml

```
---
```

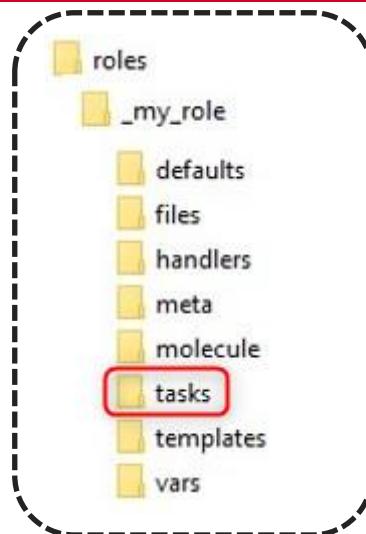
```
- name: install httpd
  yum:
    name: httpd
    state: latest

- name: start and enable httpd service
  service:
    name: httpd
    state: started
    enabled: true

- name: ensure vhost directory is present
  file:
    path: "/var/www/vhosts/{{ ansible_hostname }}"
    state: directory
```

```
- name: deliver html content
  copy:
    src: web.html
    dest: "/var/www/vhosts/{{ ansible_hostname }}"

- name: template vhost file
  template:
    src: vhost.conf.j2
    dest: /etc/httpd/conf.d/vhost.conf
    owner: root
    group: root
    mode: 0644
  notify:
    - restart_httpd
```



RÔLES TASKS (2/2)

Il est correct de mettre l'ensemble des tâches d'un rôle au sein de tasks/main.yml



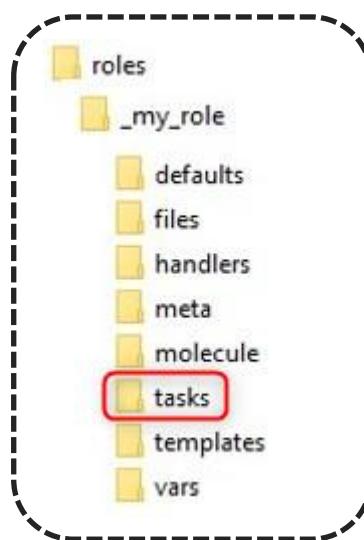
Toutefois, **pour des questions de maintenabilité**, il est fortement recommandé de **décomposer ces tâches en fichiers** que l'on importe au sein du main.yml



Code : exemple de fichier tasks/main.yml avec import

```
- import_tasks: install_httpd.yml
- import_tasks: configure_vhost.yml
- import_tasks: deliver_content.yml
- import_tasks: start_httpd.yml
```

A noter : la résolution des fichiers se fait relativement au répertoire tasks du rôle



Différence entre import_* et include_*:

Pour l'import, l'ensemble des instructions est préprocessé au moment où le playbook est parse

Pour l'include, l'ensemble des instructions est processé au moment où le playbook est exécuté



Code : exemple de fichier tasks/main.yml avec include

```
- include_tasks: '{{ ansible_os_family | lower }}/install_packages.yml'
```



RÔLES DEFAULTS ET VARS

- Ces 2 sous répertoires d'un rôle contiennent les variables utiles pour le rôle
- **defaults/main.yml** contient les **variables par défaut, surchargeables** pour un utilisateur du rôle

exemple :

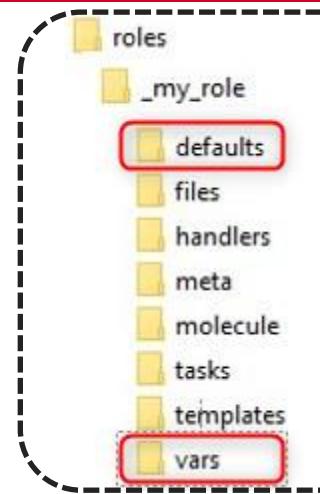
- la version d'un progiciel,
- le répertoire de déploiement d'un projet



Code : exemple de fichier defaults/main.yml

```
molecule_virtualenv_dir: ~/.venv/molecule
```

- **vars/main.yml** contient les **variables internes** au rôle qui **ne peuvent pas être surchargées sans avoir une bonne connaissance du rôle.**

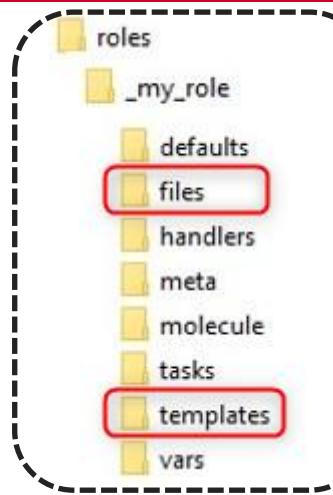


RÔLES FILES ET TEMPLATES

- Ces 2 sous répertoires d'un rôle contiennent les fichiers utiles pour le rôle
 - **files** contient les fichiers statiques nécessaires aux tâches du rôle
 - **templates** contient les modèles Jinja2 qui seront valorisées dynamiquement par les tâches du rôle



L'utilisation des **templates** Jinja2 sera abordé dans le § **Templating & Variables**



RÔLES META

- **meta/main.yml** contient :
 - les **dépendances** nécessaires au rôle
 - le nom de l'auteur,
 - le mode de licensing
 - les plateformes supportées par le rôle



Code : exemple de fichier meta/main.yml

```
galaxy_info:  
  role_name: myrole  
  author: myname  
  license: mylicense  
  min_ansible_version: 2.9  
  platforms:  
    - name: ubuntu  
      versions:  
        - 20.04  
        - 18.04  
  dependencies:  
    - role: 'mydependency1'  
    - role: 'mydependency2'
```



A noter : les rôles mydependency1 et 2 seront lancés avant _my_role



RÔLES META - PHILOSOPHIE

Avec Meta

- Vous n'avez pas nécessairement
 - l'expertise des stacks dont dépend votre rôle
 - La connaissance des policy à respecter
- Promeut
 - La modularité
 - La capitalisation de composant transverse



Sans Meta

- Lorsque vous avez une bonne maîtrise du composant à déployer
- Permet d'avoir des rôles autonomes
- Réduit la complexité de vos playbooks
- Promeut la simplicité

Tout est question d'équilibre



RÔLES HANDLERS

- **handlers/main.yml** contient les définitions des différents handlers du rôle, activés par l'instruction notify



Code : exemple de tâches déclenchant le handler

```
- name: Ensure SSH port is defined
  become: yes
  lineinfile:
    dest: '/etc/ssh/sshd_config'
    regexp: '^Port'
    line: "Port {{security_ssh_incoming_port}}"
  notify: event_security_ssh_configuration_modified
  when: security_ssh_allow_incoming_connection
```



Code : exemple de fichier handlers/main.yml

```
- name: on_security_ssh_configuration_modified
  become: yes
  systemd:
    name: ssh
    state: reloaded
  listen: event_security_ssh_configuration_modified
```



Proposition de convention de nommage
handler : on_{nom du rôle}_{nom de l'action}
notify : event __{nom du rôle}__{nom de l'action}



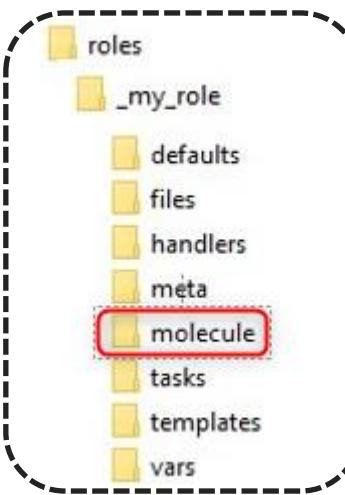
RÔLES MOLECULE

notions
avancées

- Ce répertoire contient l'ensemble des fichiers nécessaires aux **tests unitaires** du rôle s'appuyant sur le framework 'molecule'



L'utilisation des tests unitaires avec **Molecule** sera abordée dans le § Tests Unitaires



RÔLES

UTILISATION AU SEIN D'UN PLAYBOOK



Code : exemple de playbook déclenchant 2 rôles

```
- name: ansible nodes post install
hosts:
- iac_servers
roles:
- automation/ansible_post_installation
- automation/molecule_installation
```



A noter : par défaut, les rôles sont cherchés au sein du sous répertoire rôles, par rapport au playbook exécuté



Code : exemple de playbook déclenchant 1 rôle avec passage de paramètre

```
- name: molecule install
hosts:
- iac_servers
roles:
- role: automation/molecule_installation
  virtualenv_dir: ~/venv_mol
```

Idéalement, l'argument virtualenv_dir est défini dans defaults/main.yml du rôle, ce qui permet sa surcharge dans le cadre du playbook ou dans le cadre de l'inventaire



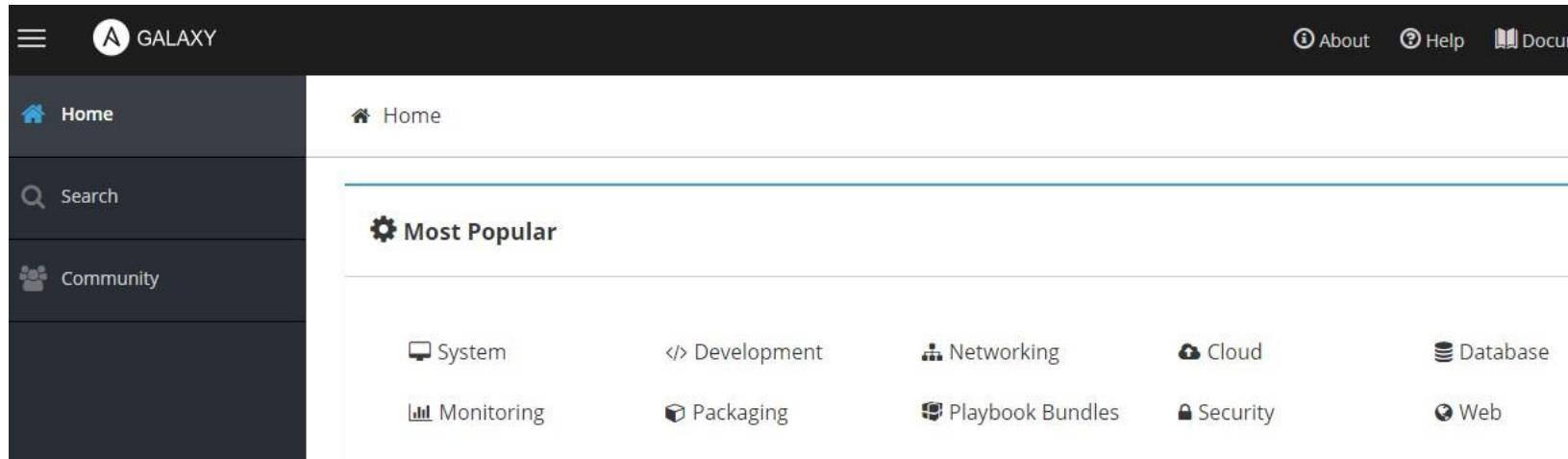
Si un rôle est déclaré plusieurs fois au sein d'un playbook, il ne sera exécuté qu'une seule fois, à moins que les paramètres du rôle ne soient différents à chaque déclaration

RÔLES

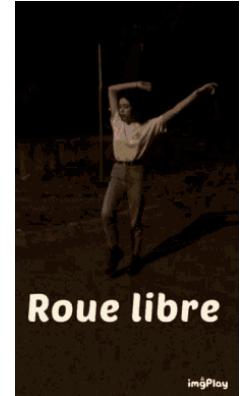
NE PAS REINVENTER LA ROUE

- Redhat met à disposition un site centralisant des rôles / collections mises à disposition par la communauté :
<https://galaxy.ansible.com/>

C'est l'équivalent de dockerhub pour Docker



The screenshot shows the Ansible Galaxy homepage. On the left is a sidebar with 'Home', 'Search', and 'Community' buttons. The main area has a 'Home' button at the top. Below it, a 'Most Popular' section is displayed with icons for System, Development, Networking, Cloud, Database, Monitoring, Packaging, Playbook Bundles, Security, and Web. To the right, there's a decorative image of a person on a trapeze with the text 'Roue libre'.



- Une fois le rôle / collection trouvé, il ne reste plus qu'à l'installer sur le nœud ansible via une commande ansible-galaxy. La commande d'installation est précisée au sein de chaque rôle / collection.



This screenshot shows a specific role page on Galaxy. It includes an 'Info' button, 'Minimum Ansible Version' (2.6), 'Installation' instructions (\$ ansible-galaxy install datadog.datadog), and a red 'IMPORTANT' stamp.



Différence entre rôle et collection :
les collections sont un nouveau concept introduit dans les dernières versions d'Ansible. Voyez-le comme un rôle ++



RÉCAPITULATIF

Si le notify n'est pas déclenché, le handler est-il exécuté ?

Qu'est-ce qu'un handler ?

Qu'est-ce qu'un rôle ?

A quoi correspondent les metas ?

Peut-on utiliser plusieurs rôles dans un playbook ?



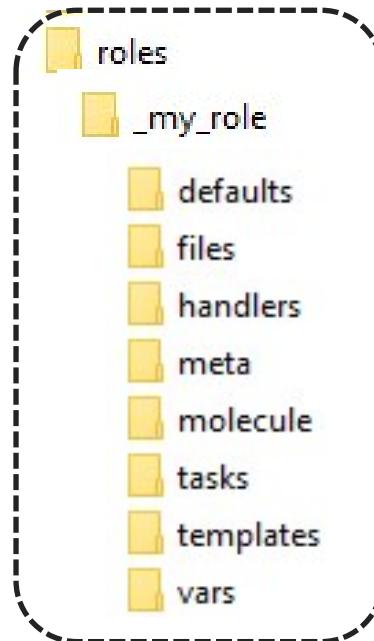
FIL ROUGE DU TP REFRACTORING → ROLE



15 min

Objectif : Refactoriser le playbook du TP3 en rôles

- Comment ferez-vous?
 - Travail en groupe de deux
- Discussion autour de vos réflexions



FIL ROUGE DU TP

REFRACTORING → ROLE



15 min

Objectif : Refactoriser le playbook du TP3 en rôles

- Comment ferez-vous?
 - Travail en groupe de deux
- Discussion autour de vos réflexions

Granularité / abstraction dans le refactoring

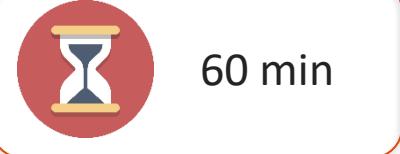
- Penser réutilisabilité
- Décorrérer avec le cycle de vie des composants

Plusieurs méthodes pour construire les rôles

- Globale (rôle installation & configuration)
- Semi découpé (rôle installation, rôle configuration)
- Très découpé (rôle installation, rôle configuration, rôle créer utilisateur, rôle tache admin ...)



FIL ROUGE DU TP REFRACTORING → ROLE



Objectif : Refactoriser le playbook du TP3 en rôles

- Au travail !
 - Refactoriser le code selon votre approche

TODO :

Lancer le playbook workshop-tp/init-tp4.yml

Modifier les éléments du dossier workshop-tp/tp4



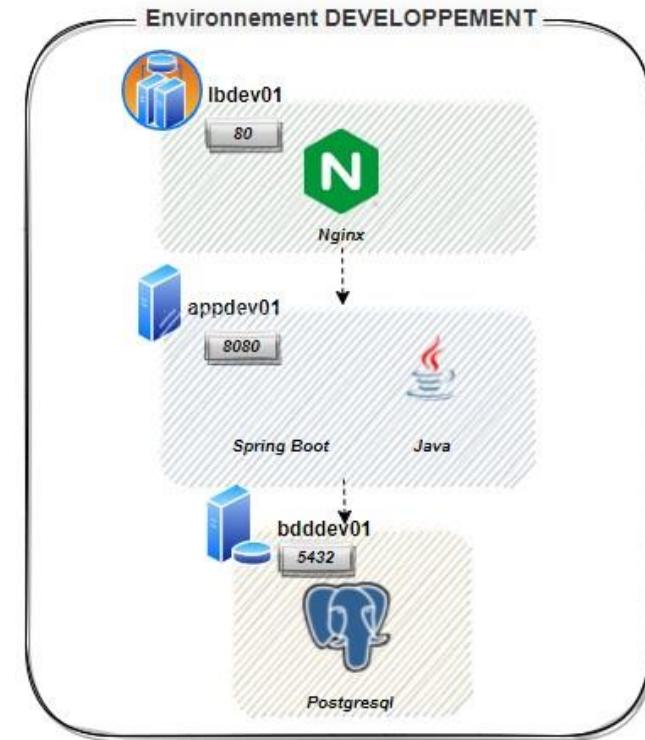
FIL ROUGE DU TP REFRACTORING → ROLE



10 min

Objectif : Refactoriser le playbook du TP3 en rôles

- Déployer la plateforme de développement avec votre refactoring



Commandes (se positionner dans le répertoire racine des TPs) :

```
ansible-playbook -i workshop-tp/inventories/rec workshop-tp/tp4/setup-petclinic-env.yml
```



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

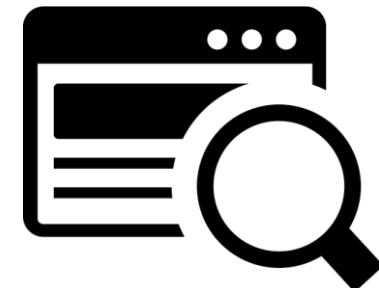
- **Via les faits relatifs aux nœuds managés** (cf. fact gathering)



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: "^\w+AllowGroups"
    line: "AllowGroups {{security_ssh_group_name}}"
    state: present
    backup: yes
```

Valeur de la variable security_ssh_group_name



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)



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
```



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

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 'I'

plusieurs opérateurs YAML existent :

I remplace les sauts de ligne par des espaces

> conserve les sauts de ligne



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

FICHIER 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)	 Code : condition jinja2 <pre>{% if variable is defined %} value of variable: {{ variable }} {% else %} variable is not defined {% endif %}</pre>  Code : boucle jinja2 <pre>{% for i in web_servers %} nom du serveur {{ i }} {% endfor %}</pre> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"><i>web_servers est une variable</i></div>
<code>{{ variable filtre1 }}</code>	Application d'un 'filtre' jinja2 à la variable	 Code : filtre <pre>{{ variable upper }}</pre>
<code>{{ variable filtre1 filtre2 filtre3 }}</code>	Chainage de filtres via piping (pour les dev, équivalent à DSL filtre1(variable).filtre2().filtre3())	 Code : filtres + piping jinja2 <pre>{{ variable default("non définie") upper }}</pre>



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 : pipping

```
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

③ On ne conserve que l'email des items filtrés



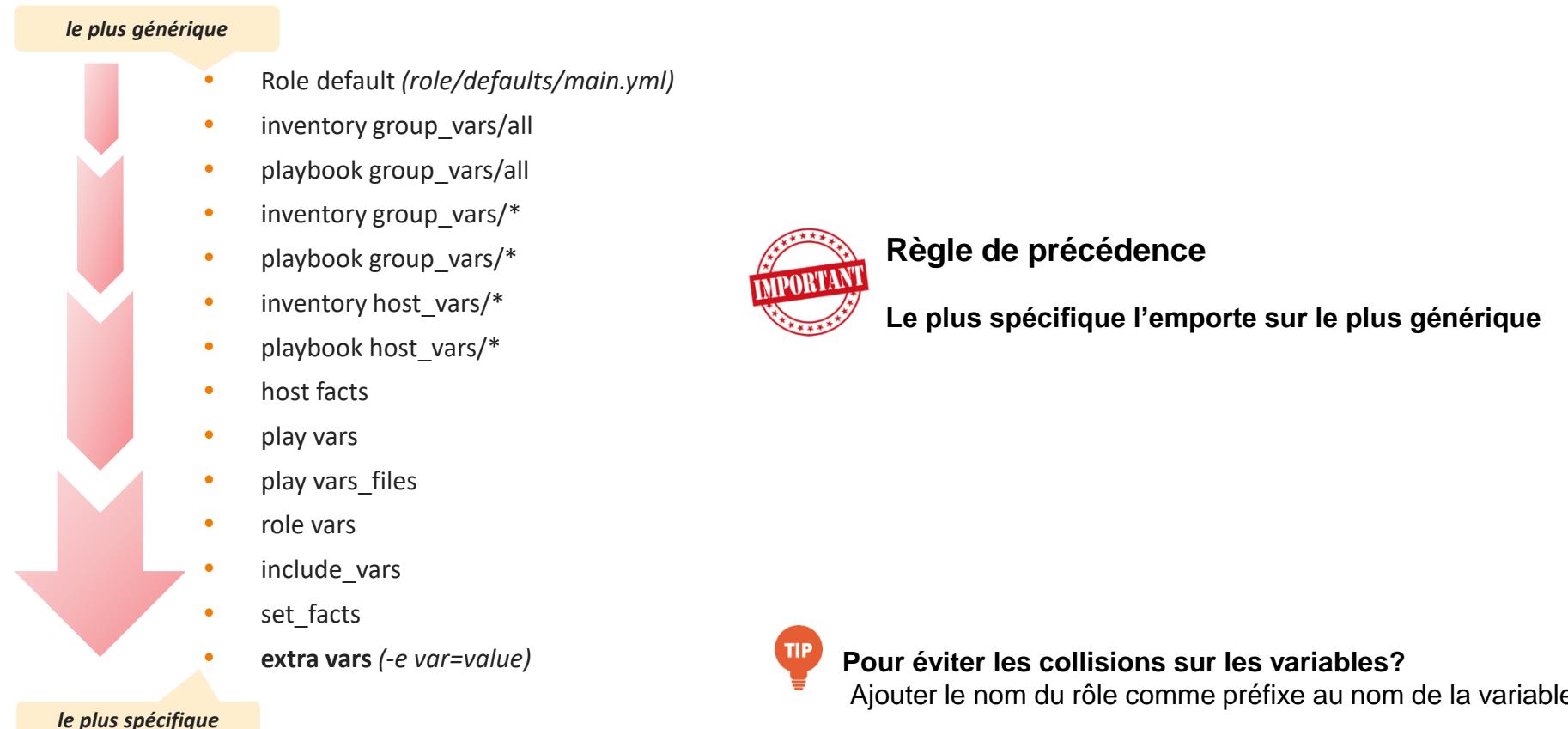
Le pipping demande de l'habitude et viendra avec l'expérience!



TEMPLATING & VARIABLES

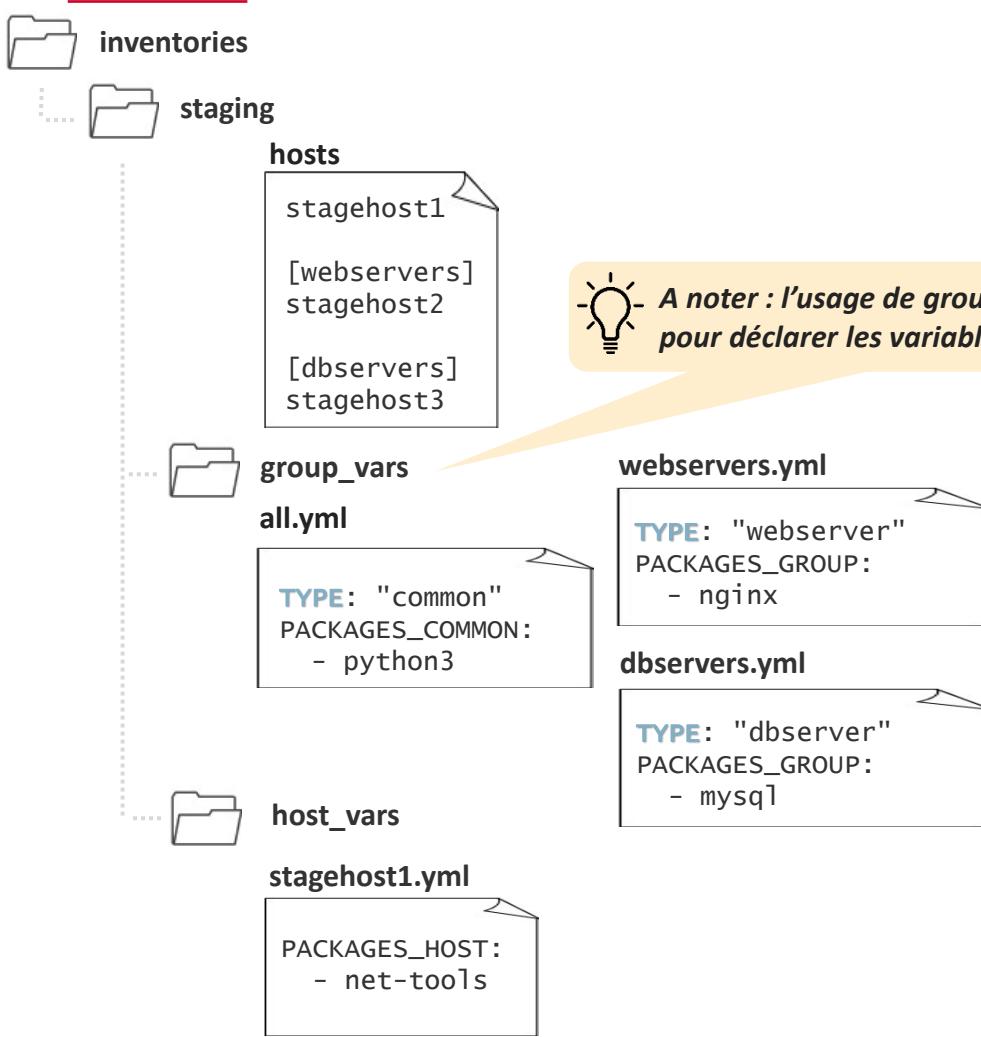
REGLE DE PRECEDENCE DES VARIABLES

- 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 précédence** ci-dessous est appliquée.



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:
  |--@dbservers:
    |--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}
  |--@webservers:
    |--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 ?



FIL ROUGE DU TP

TEMPLATING ET VARIABILISATION



15 min

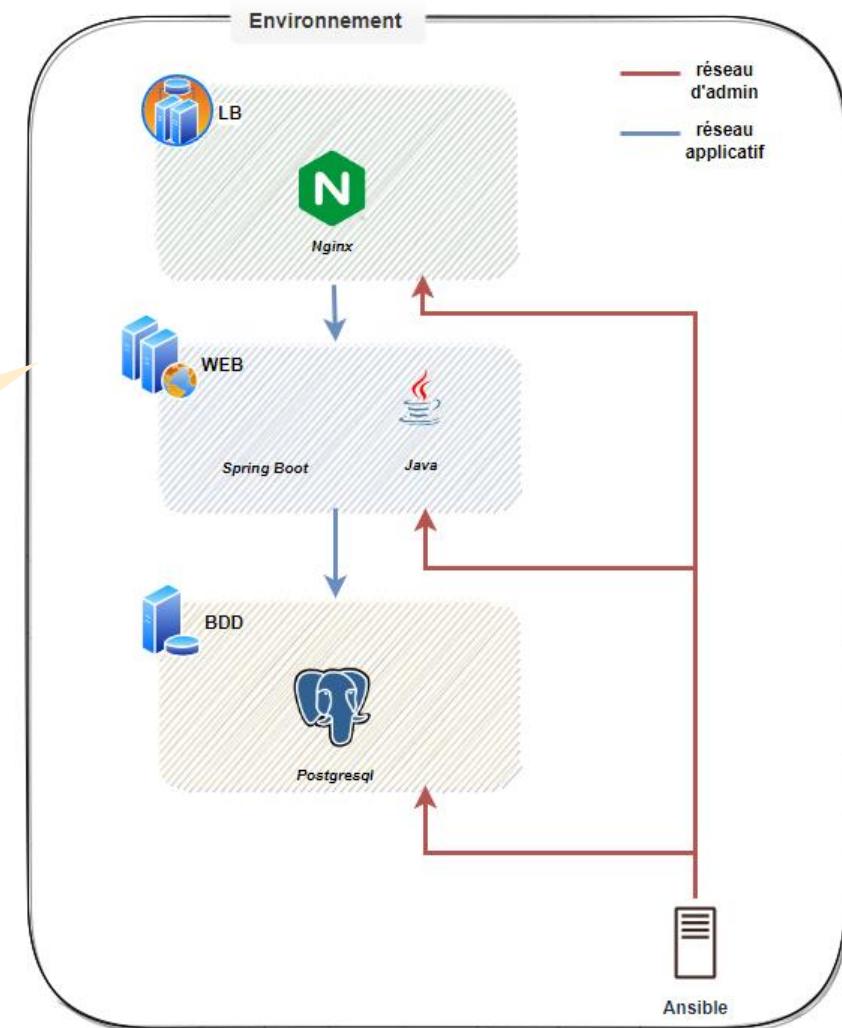
Objectif : Ajout de template et variabilisation des playbook

- Comment ferez-vous?
 - Travail en groupe de deux pendant 10 min
- Discussion autour de vos réflexions

Dans un environnement réel, les machines LB/WEB/BDD auront plusieurs cartes réseaux : l'une dédiée aux flux applicatifs, une autre dédiée aux flux d'administration (ansible)

=> autrement dit, les ips présentes dans l'inventaire ansible seront distinctes des ips présentes au niveau de la configuration applicative (conf nginx notamment)

Pour représenter cette complexité, nous allons introduire une variable `upstream_ip` définissant l'ip du serveur web pour les flux applicatifs qui sera utilisée au niveau du fichier de configuration nginx.



FIL ROUGE DU TP

TEMPLATING ET VARIABILISATION



1h30

Objectif : Ajout de template et variabilisation des playbook

- Au travail !
 - Refactoriser le code selon votre approche

TIP Utiliser ***le module debug avec l'option var*** pour afficher la structure et les valeurs d'une variable (ex. : groups, hostvars)



TODO :

Lancer le playbook workshop-tp/init-tp5.yml

Modifier les éléments du dossier workshop-tp/tp5



FIL ROUGE DU TP

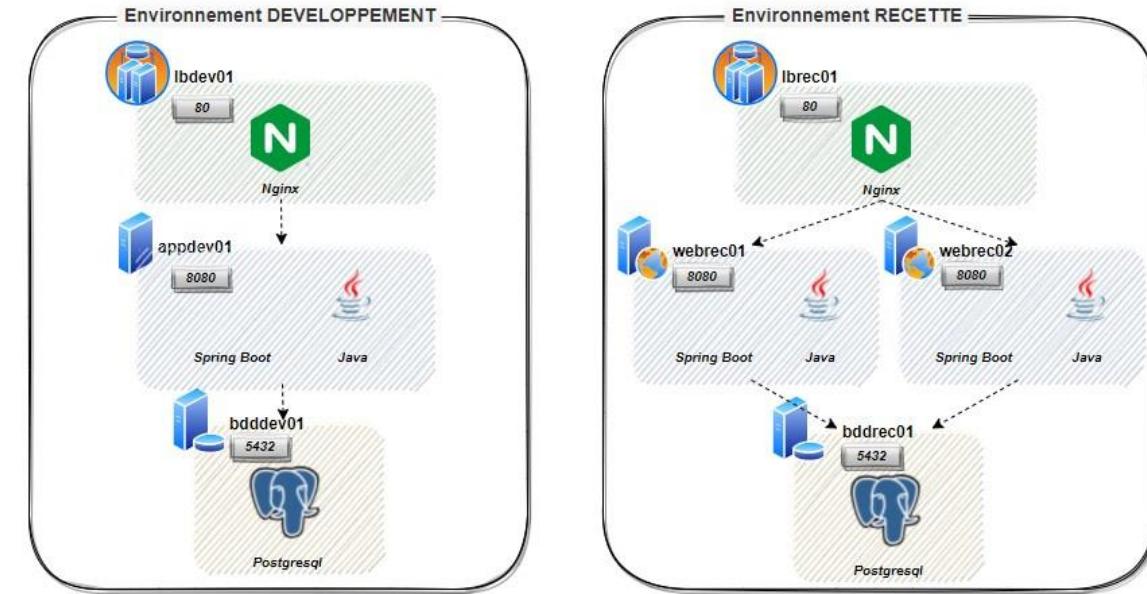
TEMPLATING ET VARIABILISATION



15 min

Objectif : Ajout de template et variabilisation des playbook

- Déployer la plateforme de recette avec votre refactoring



Commandes (se positionner dans le répertoire racine des TPs) :

```
ansible-playbook -i workshop-tp/inventories/rec workshop-tp/tp5/setup-petclinic-env.yml
```



GESTION DES ERREURS

PRESENTATION

notions
avancées

- Par défaut, tout code retour d'une commande distinct de 0 ou tout statut d'échec d'un module provoquera l'arrêt du play.
- Pour les tâches, **ignore_errors** permet d'ignorer le statut failed.



Code : ignore_errors au niveau d'une tâche

```
---
```

```
- name: do some stuff
  command: /bin/foo
ignore_errors: yes
```



Code : ignore_errors au niveau d'un playbook

```
- hosts: all
ignore_errors: yes
tasks:
- name: ...
```

- Pour les hosts, **ignore_unreachable** permet d'ignorer une machine injoignable.



Code : ignore_errors au niveau d'une tâche

```
---
```

```
- name: do some stuff
  command: /bin/foo
ignore_unreachable: yes
```



Code : ignore_errors au niveau d'un playbook

```
- hosts: all
ignore_unreachable : yes
tasks:
- name: ...
```



GESTION DES ERREURS ERREURS & HANDLERS

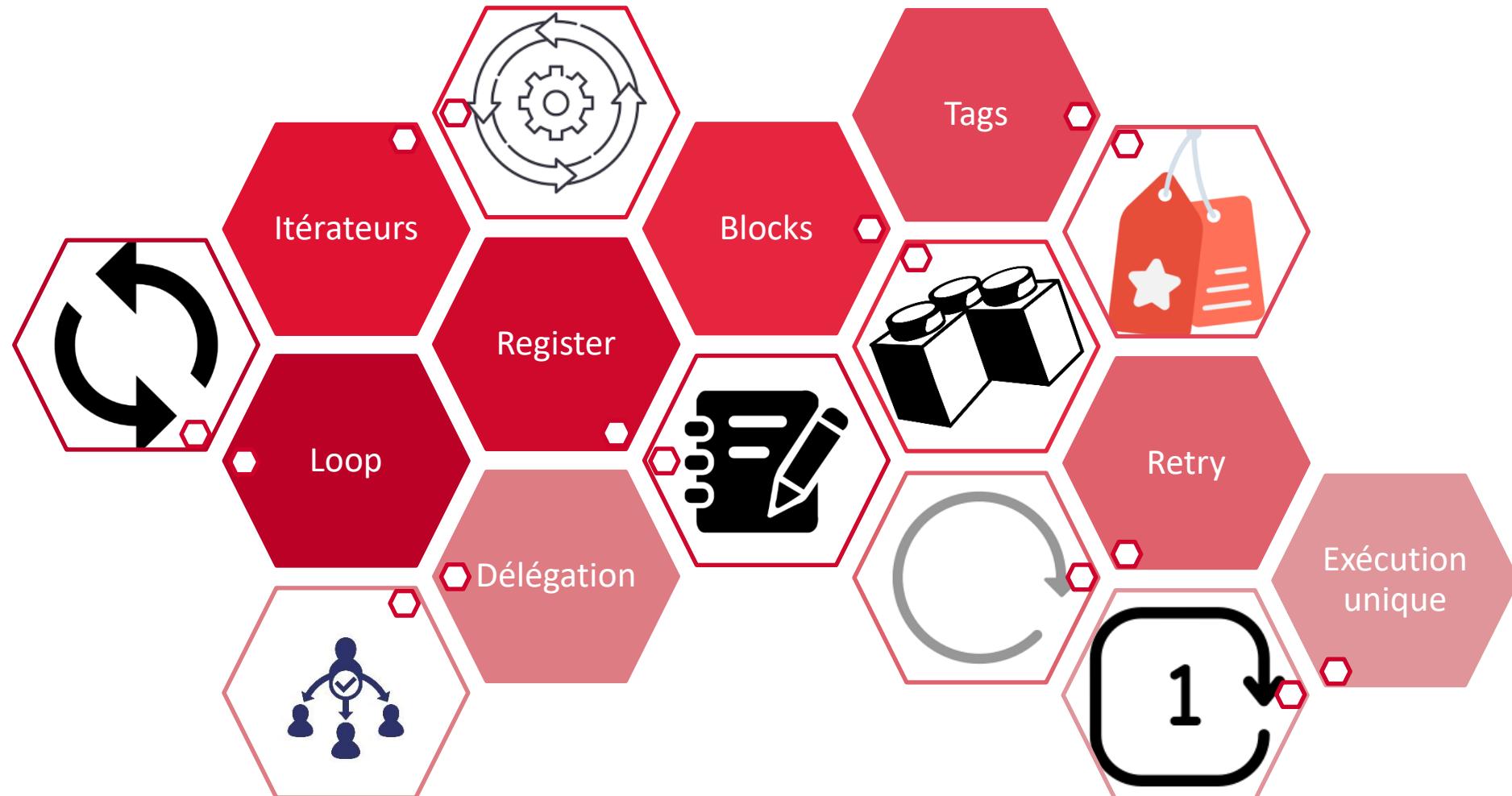
notions
expert

- Les handlers sont lancés une fois toutes les tâches réalisées.
- Dans le cas où une tâche déclenche un notify et qu'une tâche ultérieure échoue, le handler ne sera jamais exécuté ce qui peut donner lieu à un état bancal de la machine.
- Pour remédier à cette situation, il est possible de forcer l'exécution des handlers après un échec :
 - Avec l'instruction « `force_handlers: true` » déclarée au niveau du playbook
 - Via la ligne de commande `--force-handlers`



USAGES AVANCÉS

SOMMAIRE



USAGES AVANCÉS

LOOP

■ Boucle simple



Code : boucle sur une tache

```
- name: Ensure services are started
service:
  name: "{{ item }}"
  state: started
loop:
  - service1
  - service2
```



Code : boucle avec variable externalisée

```
vars:
  services:
    - service1
    - service2
- name: Ensure services are started
service:
  name: "{{ item }}"
  state: started
loop: "{{ services }}"
```



A noter : il est possible de boucler sur des structures complexes en utilisant item.monattribut



Les versions antérieures d'Ansible (<2.5) s'appuyaient sur une syntaxe différente pour réaliser les boucles à l'aide des mots clés **with_*** (**with_items**, **with_files**, **with_sequences**, ...)



USAGES AVANCÉS LOOP & REGISTER

Boucle avec register



Code : boucle sur une tache

```
- name: Display items
  shell:
    echo "{{ item }}"
  loop:
    - service1
    - service2
  register: echo_results
```



A noter : le contenu de la variable liée à register est sensiblement différent au sein d'une boucle



L'ordre dans la liste finale correspond à l'ordre défini dans le champ loop



Code : boucle avec variable externalisée

```
"echo_results": [
  {
    "changed": true,
    "start": "...",
    "end": "...",
    "delta": "...",
    "cmd": "echo \"service1\"",
    "item": "service1",
    "invocation": {
      "module_args": "echo \"service1\"",
      "module_name": "shell"
    },
    "rc": 0,
    "stdout": "service1"
    "stderr": ""
  },
  {
    "changed": true,
    "start": "...",
    "end": "...",
    "delta": "...",
    "cmd": "echo \"service2\"",
    "item": "service2",
    "invocation": {
      "module_args": "echo \"service2\"",
      "module_name": "shell"
    },
    "rc": 0,
    "stdout": "service2"
    "stderr": ""
  }
]
```



USAGES AVANCÉS

LOOP & CONDITIONS

■ Boucle avec condition



Code : boucle sur une tache

```
- name: Ensure Nginx is installed if enough space on root
  apt-get:
    name: nginx
    state: latest
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 100000000
```



A noter : la condition est testée pour chaque élément

■ Ansible parcourt l'ensemble de la liste même si certaines itérations ne sont pas exécutées du au conditionnement



USAGES AVANCÉS ITÉRATEURS (1/2)

- Possibilité d'écrire des boucles dans les tâches Ansible



Code : exemple d'un play contenant sans itérateur

```
---  
- name: install nginx  
  package:  
    name: nginx  
  
- name: install vim  
  package:  
    name: vim  
  
- name: install tomcat7  
  package:  
    name: tomcat7  
  
- name: install git  
  package:  
    name: git
```



Code : exemple d'un play contenant un itérateur

```
---  
- name: install package  
  package:  
    name: "{{ item }}"  
  with_items:  
    - nginx  
    - vim  
    - tomcat7  
    - git
```



USAGES AVANCÉS ITÉRATEURS (2/2)

- Plusieurs types:
 - **with_items**
 - tableau simple
 - item => l'entrée
 - loop.index => l'indice
 - loop.first, loop.last
 - **with_dict**
 - dictionnaire
 - item.key
 - item.value
 - **with_together**
 - Parcours simultané de plusieurs tableaux
 - item.0 => élément du premier tableau
 - item.1 => élément du second tableau
 - **with_nested**
 - parcours de la combinatoire des tableaux fournis

notions
expert

notions
expert



USAGES AVANCÉS BLOCK

- Ansible permet de regrouper plusieurs tâches au sein d'un seul bloc logique avec l'instruction **block**
- De la sorte, certaines directives (become*, when, ignore*, ...) s'appliquent à l'ensemble des tâches du bloc
- La gestion des erreurs s'applique aussi à l'ensemble du bloc grâce aux instructions **rescue** pour gérer la remédiation et **always** qui sera déclenché aussi bien lorsque le bloc termine en succès qu'en échec.



Code : #1 exemple d'un play sans bloc / remédiation

```
tasks:  
  - name: tache#1  
    debug:  
      msg: 'tache#1'  
    changed_when: yes  
    notify: event_demo_block  
  - name: tache#2 KO  
    command: /bin/false  
  
handlers:  
  - name: on event_demo_block  
    listen: event_demo_block  
    debug:  
      msg: 'handler event_demo_block'
```



Code : #2 exemple d'un play utilisant une remédiation

```
tasks:  
  - name: demonstration block et gestion d'erreur  
    block:  
      - name: tache#1  
        debug:  
          msg: 'tache#1'  
        changed_when: yes  
        notify: event_demo_block  
      - name: tache#2 KO  
        command: /bin/false  
    rescue:  
      - name: tache#remediation  
        meta: flush_handlers  
    always:  
      - name: tache#always  
        debug:  
          msg: "tache#always"  
  
handlers:  
  - name: on event_demo_block  
    listen: event_demo_block  
    debug:  
      msg: 'handler event_demo_block'
```



USAGES AVANCÉS

TAG (1/2)

- Permettent de labéliser différents types d'objet

- Rôles
- Tâches
- Imports (import_*, include_*, ...)



- Permettent de filtrer le périmètre d'exécution

- Par inclusion
- Par exclusion



Code : exécuter un playbook avec seulement les tags

```
$ ansible-playbook ... --tags "configuration,packages"
```



Code : exécuter un playbook à l'exception de qq tags

```
$ ansible-playbook ... --skip-tags "configuration,packages"
```

Code : exemple de tâches avec tag

```
tasks:  
- name: Install the servers  
  yum:  
    name:  
      - httpd  
      - memcached  
    state: present  
tags:  
- packages  
- webservers  
  
- name: Configure the service  
  template:  
    src: templates/etc/service.conf.j2  
    dest: /etc/service.conf  
tags:  
- configuration
```



Pensez à utiliser les options

--list-tags pour avoir la liste des tags d'un playbook

--list-tasks pour avoir la liste des tâches associées à un tag
(voir la doc pour plus de précision)



USAGES AVANCÉS

TAG (2/2)

- Exécuter uniquement les tâches du tag ‘install’



Code : exemple d'une ligne de commande

```
$ ansible-playbook all -i host site.yml -t install
```

- Exécuter uniquement les tâches des tags ‘install’ et ‘configure’



Code : exemple d'une ligne de commande

```
$ ansible-playbook all -i host site.yml -t install,configure
```

- Il existe des tags built-in :

- never : jamais exécuté, sauf si explicitement mentionnées via –t / --tags
- always
- tagged
- untagged



USAGES AVANCÉS

RETRY

notions
expert

- La capture d'un résultat permet de faire plusieurs tentatives sur une tâche



Code : exemple d'un play contenant un retry

```
- name: do some stuff
  command: grep -q toto /etc/passwd
  register: my_cmd_result
  until: my_cmd_result.rc == 0
  retries: 5          # number of attempts
  delay: 10           # in seconds
```



until est obligatoire. En cas d'absence de la condition d'arrêt, Ansible force retries à 1



Il existe un module (**wait_for**) qui adresse potentiellement les même use case que **retry**.
wait_for et **retry** ont toutefois qq différences notables.

wait_for	retry / until
wait_for est un module en tant que tel. Il s'applique principalement pour l'attente d'un provisioningning infra ou du reboot d'un système	Retry / until n'est pas un module à part entière. Il peut donc s'appliquer à n'importe quelle tâche.
Le message d'erreur est customisable	Il n'est pas possible d'avoir un message d'erreur customisé



USAGES AVANCÉS DÉLÉGATION

notions
avancées

- Elle permet de lancer une commande concernant une machine sur une autre machine
- Cette fonction est généralement utilisée dans deux cas de figure
 - Lors des phases de provisionnement des machines
 - Lors des déploiements à chaud (sans interruption de service)



Code : exemple contenant une délégation

```
- name: activer les alertes nagios pour les serveurs web
hosts: webservers
tasks:
  - name: activer les alertes nagios
    nagios: action=enable_alerts service=web host= {{ inventory_hostname }}
    delegate_to: my.nagios.server
```

variable built-in désignant le serveur de l'inventaire

la tâche s'exécute autant de fois qu'il y a de serveurs dans le groupe webservers, mais au lieu de s'exécuter sur chacun des serveurs, elle s'exécute sur le host my.nagios.server



USAGES AVANCÉS EXÉCUTION UNIQUE

notions
avancées

- Permet d'exécuter une tâche une seule fois sur une des machines du play



Code : exemple d'un play contenant une exécution unique

```
---
```

```
- name: forcer un timestamp unique pour un groupe de serveur
  set_fact:
    mytstamp: "{{ ansible_date_time.iso8601_basic_short }}"
  run_once: yes
```

- Si le play est configuré en 'serial', le run_once est exécuté une fois par groupe d'itération



RÉCAPITULATIF

Comment conditionne-t-on une itération d'un loop?

Quels sont les itérateurs disponibles?

Comment fonctionnent les tags?

A quoi sert le delegate_to?

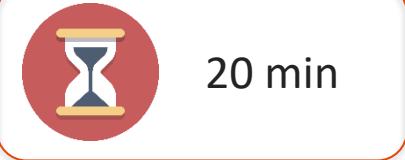
Comment fonctionne la gestion des erreurs avec un block?

Comment fait-on pour exécuter une tâche une seule fois?



FIL ROUGE DU TP

REFRACTORING → USAGES AVANCÉS



Objectif : Refactoriser le playbook du TP5 au moyen des notions avancées

- Plusieurs éléments à modifier :
 - Au niveau de la configuration du LB pour petclinic, on relance systématiquement la validation de cette dernière même si la configuration n'a pas été modifiée. Relancer cette validation est inutile. Comment géreriez-vous l'idempotence correctement?
 - Au niveau de la configuration du LB, on souhaite offrir la possibilité de désactiver le site petclinic. Définir les tâches nécessaires
- Comment feriez-vous?
 - Travail en groupe de deux (10 min)
- Discussion autour de vos réflexions (10 min)





1h30

FIL ROUGE DU TP REFRACTORING → USAGES AVANCÉS

Objectif : Refactoriser le playbook du TP5 au moyen des notions avancées

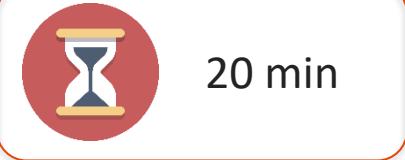
- Au travail !
 - Refactoriser le code

TODO :

- Lancer le playbook workshop-tp/init-tp6.yml
- Modifier les éléments du dossier workshop-tp/tp6
- Relancer le playbook sur l'environnement de dev



FIL ROUGE DU TP REFRACTORING → USAGES AVANCÉS



20 min

Objectif : Refactoriser le playbook du TP5 au moyen des notions avancées

- Contexte : Nous avons à l'heure actuelle la possibilité d'activer ou de désactiver un site au niveau du LB. Or ces deux fonctionnalités ne peuvent pas être actives en même temps au sein du rôle.
- Plusieurs éléments à modifier :
 - Adapter les tâches pour pouvoir déclencher explicitement les tâches d'activation et de désactivation du site.
 - Faire en sorte que la désactivation ne soit pas applicable par défaut
- Comment feriez-vous?
 - Travail en groupe de deux pendant 10 min
- Discussion autour de vos réflexions (10 min)

FIL ROUGE DU TP REFRACTORING → USAGES AVANCÉS



Objectif : Refactoriser le playbook du TP5 au moyen des notions avancées

- Au travail !
 - Refactoriser le code



TODO :

Modifier les éléments du dossier workshop-tp/tp6



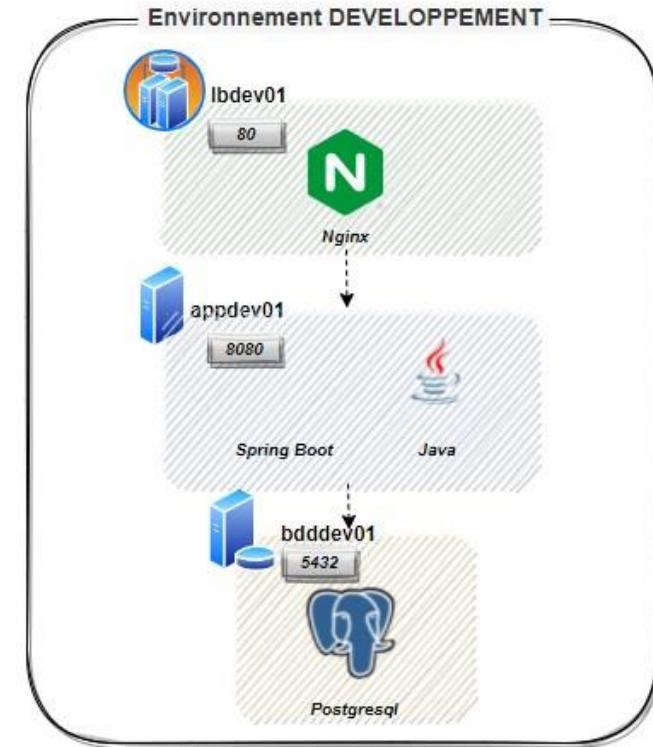
FIL ROUGE DU TP REFRACTORING → USAGES AVANCÉS



15 min

Objectif : Refactoriser le playbook du TP5 au moyen des notions avancées

- Déployer la plateforme de recette avec votre refactoring



Commandes (se positionner dans le répertoire racine des TPs) :

```
ansible-playbook -i workshop-tp/inventories/rec workshop-tp/tp6/setup-petclinic-env.yml
```



FIL ROUGE DU TP →USAGES AVANCÉS



Objectif : What if...?

- On installe plusieurs instances de petclinic sur la même machine avec des ports différents
- On installe plusieurs instances de LB avec une VIP en frontal. Le LB a besoin d'une configuration qui est générée à la volée par le script ansible. Comment fait-on?



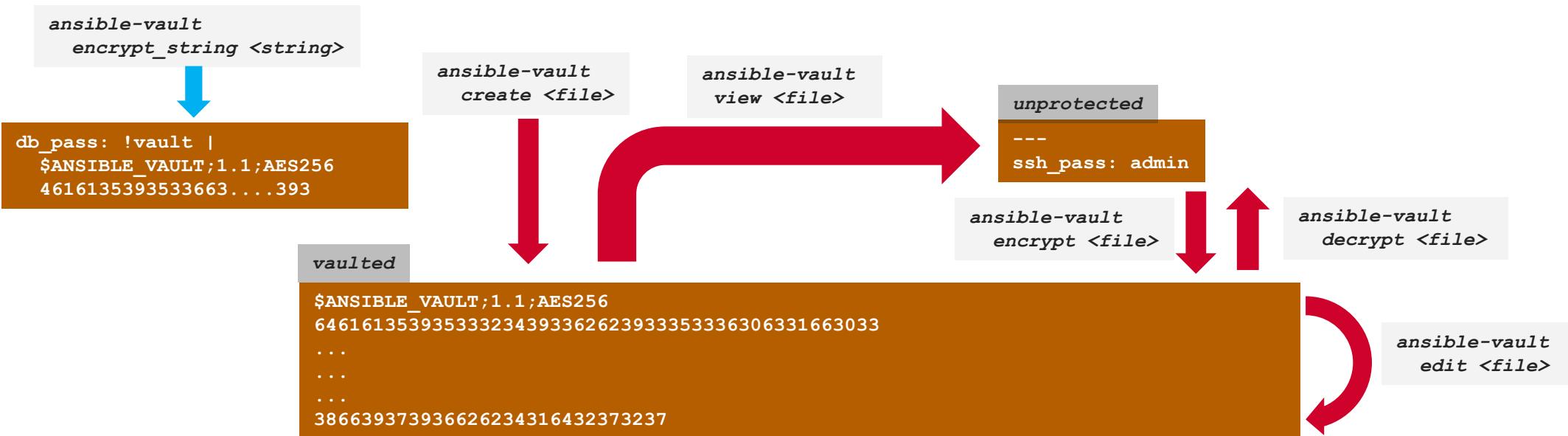
ANSIBLE VAULT

PRESENTATION

- Les vaults (coffres-forts) sont des fichiers chiffrés de variables protégés par un mot de passe
- Ils permettent de stocker des informations sensibles
 - Mots de passe
 - Clés privées
 - Tokens
- Ils sont manipulés à l'aide des commandes ansible-vault



Si votre playbook utilise plusieurs vaults, utilisez l'option --vault-id



ANSIBLE VAULT

VAULT ID

A noter : plusieurs vault id peuvent être utilisés au sein d'un playbook



Code : fichiers de password des différents vaults

```
$ echo -n bdd_vault_password > bdd_vault_password_file  
$ echo -n cert_vault_password > cert_vault_password_file
```

2 vaults ayant chacun leur password :
un pour la bdd et un pour les certificats



Code : fichier de variables non protégées

```
$ echo var_bdd_password: pA$$w0rd > bdd.yml
```

Le password utilisé pour se connecter à la BDD



Code : chiffrement du fichier de variable avec le vault id bdd, en utilisant le password du vault contenu dans le fichier bdd_vault_password_file

```
$ ansible-vault encrypt bdd.yml --vault-id bdd@bdd_vault_password_file --output bdd_vault.yml
```

A noter : si output non renseigné; bdd.yml sera chiffré

```
$ANSIBLE_VAULT;1.2;AES256;bdd  
64323665643830656635623266353330386437346436643631393365346166363032623139306536  
...  
36353438393865373235663232376239316435363034353966663839343839386339
```

A noter : @ précise l'emplacement du password



si erreur [WARNING]: Error in vault password file loading If this is not a script, remove the executable bit from the file.

Et que chmod -x ne fonctionne pas,

Remplacer le password du vault par :

```
#!/bin/bash  
echo <password>
```



ANSIBLE VAULT COMMANDES

- Passer un fichier vault.yml (contenant les valeurs chiffrées) en ligne de commande avec -e et --ask-vault-pass

A noter : @ précise qu'il s'agit d'un fichier



Code : usage d'un fichier vault.yml à l'exécution d'un playbook

```
$ ansible-playbook all -i inventories/rec site.yml -e @vault.yml --ask-vault-pass
```

- Passer un fichier vault.yml en ligne de commande avec un fichier contenant le mot de passe



Code : usage d'un fichier vault.yml à l'exécution d'un playbook

```
$ ansible-playbook all -i inventories/rec site.yml -e @vault.yml --vault-password-file .vault.pass
```

- Passer des vaults Id en ligne de commande avec des fichiers contenant le mot de passe de chaque vault



Code : usage de multiples vault (id : bdd, cert) à l'exécution d'un playbook

A noter : prompt demandera la saisie du mot de passe pour le vault id cert

```
$ ansible-playbook all -i inventories/rec site.yml -vault-id bdd@rec/bdd.pwd -vault-id cert@prompt
```

A noter : pour le vault id bdd, le mot de passe du vault est défini dans le fichier rec/bdd.pwd



FIL ROUGE DU TP

UTILISATION DU VAULT



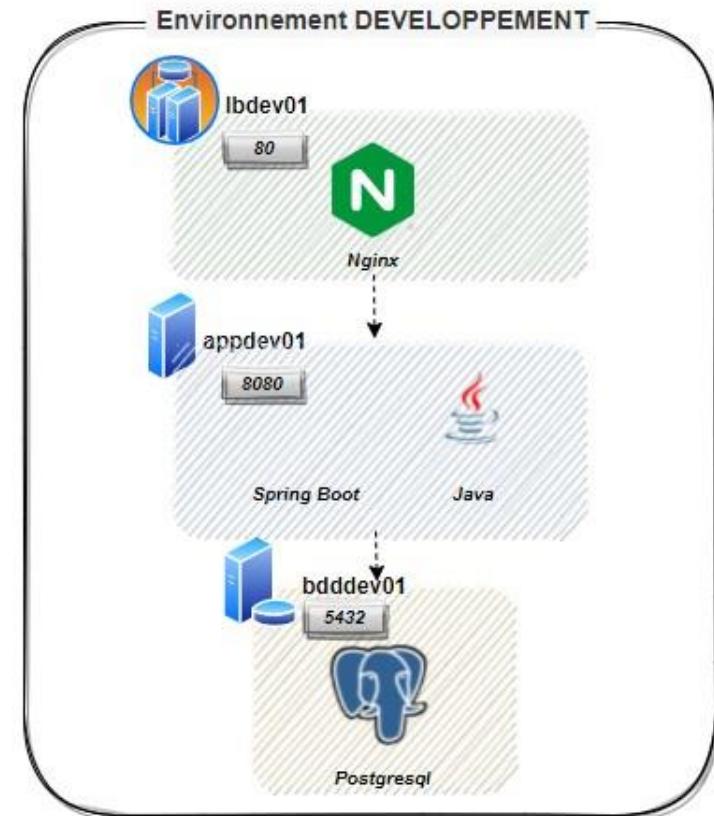
30 min

Objectif : Externaliser le mot de passe de la base de données dans un vault

- Définir un répertoire secrets par environnement au sein de vos inventaires qui contiendra les passwords de vos vaults ansible
- Créer un fichier vault par connexion à la bdd (admin, petclinic) et chiffrer les mots de passe
- Impacter vos fichiers yaml pour utiliser vos vaults
- Vérifier le bon fonctionnement de vos scripts

TODO :

- Lancer le playbook workshop-tp/init-tp7.yml
- Modifier les éléments du dossier workshop-tp/tp7
- Tester le playbook sur l'environnement de dev



TESTS AVEC ANSIBLE

GENERALITÉS

- Tester des scripts ansible peut être fastidieux.

Certains **modules/rôles modifiant l'hôte** nécessitent de **repartir d'un hôte vierge** pour s'assurer du bon fonctionnement dans le cadre d'un développement itératif

- Pour ce faire, différents niveaux de tests sont envisageables :

- Le mode **dry run**, fonctionnalité intégrée au sein des principaux modules ansible
- Les **tests unitaires** avec **molécule**
- Les **tests d'intégration** avec **vagrant** ou via des **VM provisionables**



DRY RUN AVEC ANSIBLE

GENERALITÉS

- Ansible propose 2 modes de validation de tâches qui peuvent être utilisés conjointement :
 - Le **mode check** **n'applique pas** les modifications mais **signale les modifications** que les modules auraient apportées. Les modules ne prenant pas en charge le mode check ne rapportent rien et ne font rien.



Code : exécution d'un playbook avec le mode check

```
$ ansible-playbook myplaybook.yml --check
```

- Le **mode diff applique** les modifications et **fournit des comparaisons avant/après** exécution des modules. Du fait du caractère extrêmement détaillé de ce mode, il est préférable de ne l'exécuter que sur un hôte à la fois.



Code : exécution d'un playbook avec le mode diff

```
$ ansible-playbook myplaybook.yml --diff
```



L'**utilisation conjointe du mode diff et du mode check** permet de restituer les comparaisons avant/après sans exécuter les modules sur les hôtes.



TESTS UNITAIRES AVEC MOLECULE

GENERALITÉS

notions
avancées

- Molecule est un framework pour réaliser des **tests unitaires** avec Ansible
- Molecule s'appuie sur des **images docker, podman ou des box vagrant** pour exécuter les tests
- Molecule s'exécute au sein **d'un environnement virtuel python**, à préciser au moment de l'installation de molecule. Cet environnement virtuel doit être activé avant d'utiliser molecule.



Dans la suite de la formation, nous supposerons que **l'environnement virtuel** est situé sous `~/.venv/molecule`



Code : activation de l'environnement virtuel associé à molecule

```
$ source ~/.venv/molecule/bin/activate
```



Code : désactivation de l'environnement virtuel associé à molecule

```
$ deactivate
```



TESTS UNITAIRES AVEC MOLECULE STRUCTURE DE L'ARBORESCENCE

- Molecule s'attend à une arborescence particulière sous le rôle à tester.
Cette arborescence peut être initialisée au sein de l'environnement virtuel molecule par différentes commandes.



~~Code : initialisation à partir d'un rôle inexistant~~

```
$ molecule init role _my_role --driver-name vagrant  
$ molecule init role _my_role --driver-name docker  
$ molecule init role _my_role --driver-name podman
```

deprecated
Molecule v6



~~Code : initialisation à partir d'un rôle existant~~

```
$ molecule init scenario -r _my_role --driver-name vagrant  
$ molecule init scenario -r _my_role --driver-name docker  
$ molecule init scenario -r _my_role --driver-name podman
```

deprecated
Molecule v6

```
$ molecule init scenario _my_scenario
```

Molecule v6



A noter : toute **molecule** nécessite d'avoir activé au préalable **l'env. virtuel** avec **activate**

▼ molecule

 ▼ default

 le nom du scénario de test

 group_vars

 les éventuelles surcharges de variables

 converge.yml

 molecule.yml

 prepare.yml

 verify.yml

} les fichiers molecule correspondant à différentes étapes pour le test



TESTS UNITAIRES AVEC MOLECULE USAGE

- Durant un test, molecule va traverser différentes étapes.
- La liste de ces étapes est précisée par la commande matrix



Code : liste des étapes pour la séquence test

```
$ molecule matrix test
```



Code : liste des étapes pour la séquence converge

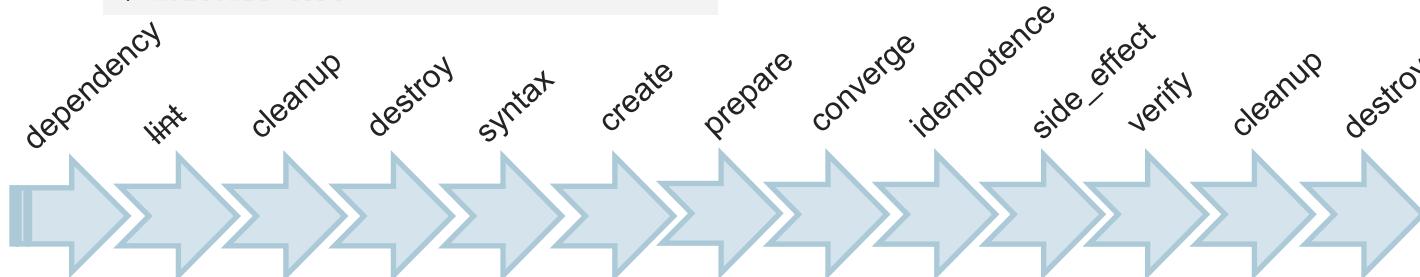
```
$ molecule matrix converge
```

- La séquence **test** est la plus complète



Code : exécuter la séquence de test sur le rôle _my_role

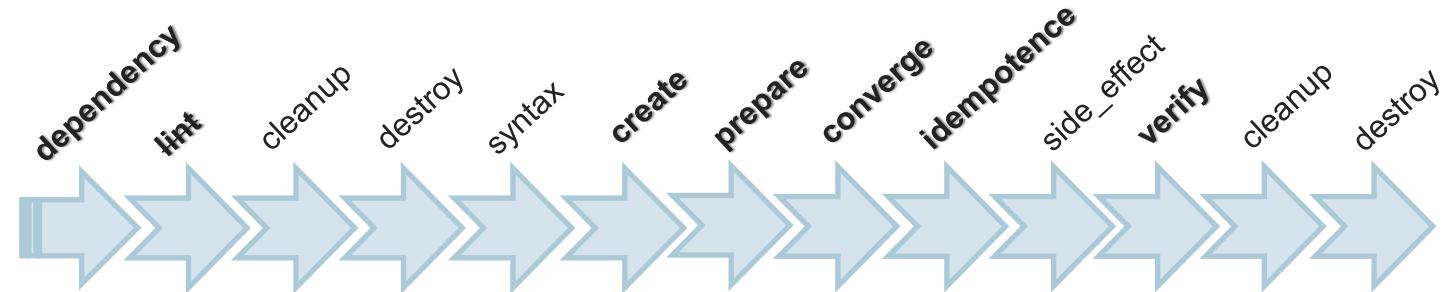
```
$ cd roles/_my_role
$ source ~/.venv/molecule/bin/activate
$ molecule test
```



A noter : enchainement des étapes pour la séquence test (ce que vous avez avec la commande matrix).

TESTS UNITAIRES AVEC MOLECULE

PRINCIPALES ETAPES



- Step **dependency** : installe les éventuelles dépendances ansible-galaxy nécessaires à l'exécution du rôle.
- Step **lint** : s'assure de l'absence de violation par rapport aux règles qualité (ansible-lint ou autre)
- Step **create** : création des hôtes par le driver pour le test.
Configuré par le fichier **molecule.yml**, via la balise platforms
- Step **prepare** : exécution d'opérations supplémentaires post création des hôtes.
Configuré par le fichier **prepare.yml**
- Step **converge** : exécution du rôle à exécuter.
Configuré par le fichier **converge.yml**
- Step **idempotence** : réexécution du rôle pour s'assurer de son idempotence
- Step **verify** : exécution des tests sur les hôtes créés
Configuré par le fichier **verify.yml**



TESTS UNITAIRES AVEC MOLECULE

REMARQUES

- Chaque étape peut être lancée individuellement.
- Molecule conserve une notion d'état pour chaque scenario et chaque hôte, de sorte à ne rejouer que ce qui est utile.



Code : état pour les hôtes et scenario

```
$ molecule list
```

Instance Name	Driver Name	Provisioner Name	Scenario Name	Created	Converged
ubuntu1804	docker	ansible	default	false	false
ubuntu2004	docker	ansible	default	false	false



état '*converged*' : affecté par les étapes *converge* et *cleanup*

- Il est possible de se connecter à l'hôte pour investiguer.



Code : se connecter à l'hôte *ubuntu2004*

```
$ molecule login --host ubuntu2004
```



état '*created*' : affecté par les étapes *create* et *destroy*

TESTS UNITAIRES AVEC MOLECULE

EXEMPLE FICHIER MOLECULE.YML

- Le fichier molecule.yml précise notamment les images de tests cible



Code : exemple molecule.yml

```
---
```

```
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name:      ubuntu1804
    image:     gearlingguy/docker-ubuntu1804-ansible
    pre_build_image: true
  - name:      ubuntu2004
    image:     gearlingguy/docker-ubuntu2004-ansible
    pre_build_image: true
  - name:      centos7
    image:     gearlingguy/docker-centos7-ansible
    pre_build_image: true

provisioner:
  name: ansible
verifier:
  name: ansible

lint:
  - set
  - ansible-lint
```



la création des hôtes s'appuie sur le driver docker



3 hôtes définis via platforms, sur lesquels seront exécutés les tests



la commande ansible-lint à lancer pour la phase de linting



geerlingguy met à disposition des images docker prêt à l'emploi pour ansible



TESTS UNITAIRES AVEC MOLECULE

EXEMPLE FICHIER PREPARE.YML

- Les images mise à disposition ne sont pas toujours complètes.
- L'étape Prepare permet d'appliquer des modifications à l'environnement de test avant l'exécution des tests proprement dit.



Code : exemple prepare.yml

```
---
```

```
- name: Prepare
  hosts: all
  gather_facts: true
  tasks:
    - name: Ensure gpg-agent is installed
      apt:
        name: gpg-agent
        state: present
        update_cache: true
        when: ansible_facts['os_family'] == "Debian"
```



Vous remarquez qu'il s'agit d'un playbook



THANKS CAPTAIN OBVIOUS

TESTS UNITAIRES AVEC MOLECULE

EXEMPLE FICHIER CONVERGE.YML

- Le fichier converge lance le rôle sur les images cible



Code : exemple converge.yml

```
---
```

```
- name: Converge
  hosts: all
  vars:
    my_var: 'my_value'
  roles:
    - role: _my_role
```



Vous remarquez qu'il s'agit d'un playbook



TESTS UNITAIRES AVEC MOLECULE

EXEMPLE FICHIER VERIFY.YML

- Le fichier verify contient les tests à exécuter



Code : exemple verify.yml

```
name: Verify
hosts: all
gather_facts: false
tasks:

  - name: Retrieve Python3 version
    shell: python3 --version
    register: python_version

  - name: Ensure Python3 version '{{python_version.stdout}}' is correct
    assert:
      that:
        - (python_version.stdout | lower) is regex("python 3\.")
    msg: |
      Python major version has to be equal to 3.
      Version output '{{python_version.stdout}}'

  - name: Retrieve Pip version
    shell: pip3 --version
    register: pip_version
    # pip_version = pip xx.xx.xx from xx/lib/python3.xx/site-packages/pip (python 3.xx)

  - name: Ensure Pip3 '{{pip_version.stdout}}' uses Python3
    assert:
      that:
        - (pip_version.stdout | lower) is regex("python 3\.")
    msg: |
      Pip3 has to used Python3.
      Version output '{{pip_version.stdout}}'
```



Vous remarquez qu'il s'agit d'un playbook



FIL ROUGE DU TP

TESTS UNITAIRES AVEC MOLECULE



5 min

Objectif : Tester ses playbooks avec Molecule

- Générer la structure de l'arborescence des tests unitaires pour le loadbalancer NGINX

Commandes :

- Activation de l'environnement molecule
source ~/.venv/molecule/bin/activate

- Se placer dans le répertoire du rôle :
`/home/ansible/workspace/workshop-tp/tp8/roles/{}_my_role}`

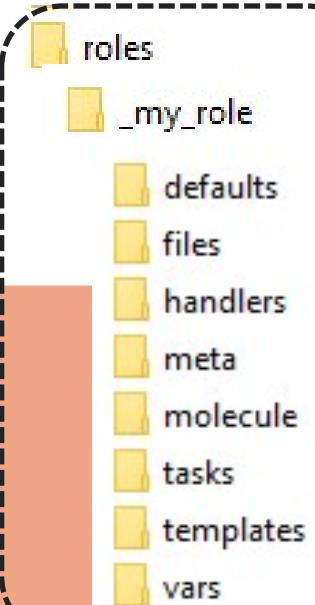
- Générer la structure molecule avec podman pour un role _my_role

~~`molecule init role academy.myrole --driver-name podman` (deprecated en molecule 6.0)~~
molecule init scenario {scenario_name} (définit une structure minimale, privilégier la copie d'un répertoire molécule existant – voir correction)

- Copier le répertoire molecule de myrole dans le role nginx
Se placer dans le répertoire du rôle nginx

TODO :

Lancer le playbook `workshop-tp/init-tp8.yml`
Modifier les éléments du dossier `workshop-tp/tp8`





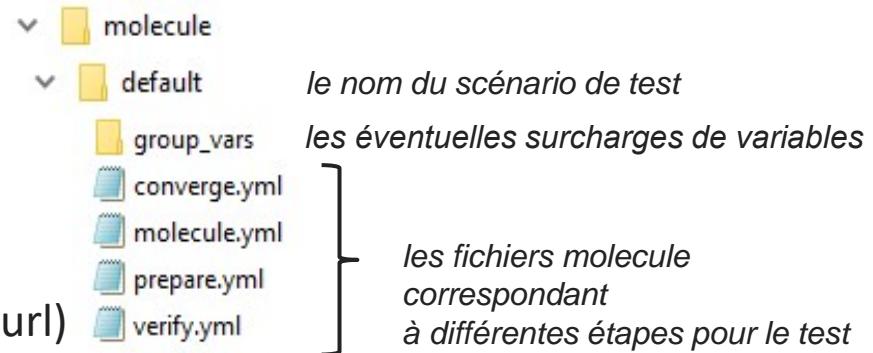
15 min

FIL ROUGE DU TP

TESTS UNITAIRES AVEC MOLECULE

Objectif : Tester ses playbooks avec Molecule

- Mettre en place un test basique pour le loadbalancer NGINX :
 - Vérifier la version du NGINX
 - Optionnel : Vérifier que le NGINX répond sur le port attendu (utiliser curl)



Actions :

- Modifier le fichier molecule pour référencer les VMs de test
 - ubuntu2004, image: geerlingguy/docker-ubuntu2004-ansible
 - debian11, image: geerlingguy/docker-debian11-ansible
- Modifier le fichier converge pour jouer le role platform/nginx
- Modifier le fichier verify pour intégrer les 2 tests nginx
- Ajouter les variables nécessaires aux tests via group_vars

/!\ lorsque les VM de tests sont lancées au sein d'une image, qq ajustements sont nécessaires :

- Lorsque vous utiliser des services, utilisez command: \${MOLECULE_DOCKER_COMMAND:-"sleep infinity"}
- Penser à regarder certains uses cases spécifiques (systemd, container inside container, ...)
cf. <https://ansible.readthedocs.io/projects/molecule/examples/>



FIL ROUGE DU TP

TESTS UNITAIRES AVEC MOLECULE



10 min

Objectif : Tester ses playbooks avec Molecule

- Lancer le cycle de tests en mode itératif (pour chaque phase et pour l'ensemble des phases)

Commandes :

- Lister l'état des VM de test

```
molecule list
```

- Lancer l'étape converge et lister les états résultants

```
molecule converge #execution du role  
molecule list
```

- Lancer l'étape verify ; ajuster les scripts en fonction des résultats

```
molecule verify
```

- Récréer l'env de test from scratch et lancer un test complet

```
molecule destroy
```

```
molecule list
```

```
molecule test
```



TESTS D'INTÉGRATION AVEC VAGRANT

GÉNÉRALITÉS (1/3)

notions
expert



Votre OS doit avoir le support de la virtualisation activée (paramètre du bios)
Si vous faites tourner vagrant dans une VM, il faut activer le support de la virtualisation imbriquée (VT-x / AMD-v imbriqué)

- Si vous n'avez pas d'environnement d'intégration instanciable à la demande, Vagrant permet de piloter la création de VM (ex. via virtualbox) sur son poste de travail
- Il permet ainsi de gérer un écosystème de VM sur lesquelles exécuter ses playbooks ansible
- Vagrant s'appuie sur un fichier de configuration des VM à instancier
- Les principales commandes vagrant sont détaillées ci-dessous :



Code : démarrer les VM

```
$ vagrant up
```



Code : démarrer la VM my_vm

```
$ vagrant up my_vm
```



Code : se connecter en ssh à la VM my_vm

```
$ vagrant ssh my_vm
```



Code : suspendre les VM

```
$ vagrant suspend
```



Code : détruire les VM

```
$ vagrant destroy
```



Utiliser --debug dans les commandes vagrant pour avoir plus de traces



TESTS D'INTÉGRATION AVEC VAGRANT GÉNÉRALITÉS (2/3)



Code : Configuration Vagrant pour une VM demobdd01

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure('2') do |config|  
  
    config.ssh.insert_key = false
    config.ssh.verify_host_key = :never
    config.ssh.forward_agent = true  
  
    config.vm.define 'demobdd01' do |demobdd01|
        demobdd01.vm.hostname = 'demobdd01'
        demobdd01.vm.box = 'ubuntu/focal64'
        demobdd01.vm.allow_hosts_modification = true
        demobdd01.vm.network 'private_network', ip: '172.28.128.11', hostname: true
        demobdd01.vm.network 'forwarded_port', id: 'ssh', guest: 22, host: 2211
        demobdd01.vm.network 'forwarded_port', id: 'http', guest: 90, host: 90
        demobdd01.vm.provider 'virtualbox' do |vb|
            vb.name = demobdd01.vm.hostname
            vb.memory = 2048
            vb.cpus = 2
        end
        demobdd01.vm.provision 'ansible_local' do |ansible|
            ansible.raw_arguments = ['--connection=local']
            ansible.verbose = 'vv'
            ansible.playbook = 'playbooks/integ/init_bdd.yml'
            ansible.inventory_path = 'inventories/integ/local/inventory.yml'
        end
        demobdd01.vm.post_up_message = '*** VM ready ***'
    end
    # configuration d'un autre VM à la suite
end
```

I'image vagrant de la box à utiliser, accessible via le catalogue <https://app.vagrantup.com/boxes/search>

La configuration réseau de la VM. Il est indispensable d'exporter les ports qui sont accessibles depuis l'extérieur de la VM (comme pour docker)

les caractéristiques de la VM

le playbook ansible pour initialiser la VM (ansible sera installé sur la VM, d'où la connexion local)



TESTS D'INTÉGRATION AVEC VAGRANT

GÉNÉRALITÉS (3/3)



Code : 3 moyens pour se connecter en ssh à la VM

```
$ vagrant ssh demobdd01  
$ ssh -i .vagrant.d/insecure_private_key -p 2211 vagrant@127.0.0.1  
$ ssh -i .vagrant.d/insecure_private_key -p 22 vagrant@172.28.128.11
```



Cf conf vagrant :

```
demobdd01.vm.network 'private_network', ip: '172.28.128.11', hostname: true  
demobdd01.vm.network 'forwarded_port', id: 'ssh', guest: 22, host: 2211
```



Pour spécifier le fichier de conf vagrant (sous windows)
set VAGRANT_VAGRANTFILE=playbooks\integ\my_vagrantfile



Pour rendre les logs ansible lisibles (sous windows)
set ANSIBLE_STDOUT_CALLBACK=debug



DEMO AVEC VAGRANT SOUS WINDOWS (1/3)

Temps : 30 mn



Attention à la compatibilité entre VirtualBox et Vagrant. Au moment de la rédaction du support, la dernière version de virtualbox (7.0) n'était pas supportée par Vagrant (2.3.2)



Code : Télécharger la box vagrant pour vos VM

```
$ vagrant box add ubuntu/focal64
```

A ne faire qu'une fois

```
==> box: Loading metadata for box 'ubuntu/focal64'  
    box: URL: https://vagrantcloud.com/ubuntu/focal64  
==> box: Adding box 'ubuntu/focal64' (v20221103.0.0) for provider: virtualbox  
    box: Downloading: https://vagrantcloud.com/ubuntu/boxes/focal64/versions/20221103.0.0/providers/virtualbox.box  
Download redirected to host: cloud-images.ubuntu.com  
    box:  
==> box: Successfully added box 'ubuntu/focal64' (v20221103.0.0) for 'virtualbox'!
```



Code : Démarrer la VM lbdev01

```
$ cd <racine du projet>  
$ set ANSIBLE_STDOUT_CALLBACK=debug  
$ set VAGRANT_VAGRANTFILE=workshop-answers\tp9\vagrantfile_windows  
$ vagrant up lbdev01
```

L'exposition des ports définie au sein du fichier de config vagrant

Le répertoire du projet est accessible sous /vagrant au sein de la VM

Pour rejouer les scripts ansible au sein de la VM, ajouter à la commande ansible-playbook ... --connection=local -limit ...

```
Bringing machine 'lbdev01' up with 'virtualbox' provider...  
==> lbdev01: Importing base box 'ubuntu/focal64'...  
==> lbdev01: Matching MAC address for NAT networking...  
==> lbdev01: Checking if box 'ubuntu/focal64' version '20221103.0.0' is up to date...  
==> lbdev01: Setting the name of the VM: lbdev01  
==> lbdev01: Clearing any previously set network interfaces...  
==> lbdev01: Preparing network interfaces based on configuration...  
    lbdev01: Adapter 1: nat  
    lbdev01: Adapter 2: hostonly  
==> lbdev01: Forwarding ports...  
    lbdev01: 22 (guest) => 2211 (host) (adapter 1)  
    lbdev01: 90 (guest) => 90 (host) (adapter 1)  
==> lbdev01: Running 'pre-boot' VM customizations...  
==> lbdev01: Booting VM...  
==> lbdev01: Waiting for machine to boot. This may take a few minutes...  
    lbdev01: SSH address: 127.0.0.1:2211  
    lbdev01: SSH username: vagrant  
    lbdev01: SSH auth method: private key  
    lbdev01: Warning: Connection reset. Retrying...  
==> lbdev01: Machine booted and ready!  
==> lbdev01: Checking for guest additions in VM...  
==> lbdev01: Setting hostname...  
==> lbdev01: Configuring and enabling network interfaces...  
==> lbdev01: Mounting shared folders...  
    lbdev01: /vagrant => C:/_SSG_labs/_projects/_academy/_ansible_initiation  
==> lbdev01: Running provisioner: ansible_local...  
    lbdev01: Installing Ansible...
```



DEMO AVEC VAGRANT SOUS WINDOWS (2/3)

```
lbdev01: Running ansible-playbook...
cd /vagrant && PYTHONUNBUFFERED=1 ANSIBLE_NOCOLOR=true ansible-playbook --limit="lbdev01" --
inventory-file=workshop-answers/inventories/_vagrant/inventory.yml -vv --connection=local
workshop-answers/tp9/setup-petclinic-env.yml
ansible-playbook [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/vagrant/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/vagrant/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible-playbook
  python version = 3.8.10 (default, Jun 22 2022, 20:18:18) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
Using /etc/ansible/ansible.cfg as config file
statically imported: /vagrant/workshop-answers/tp6/roles/platform/postgres-15/tasks/_install.yml
...
PLAYBOOK: setup-petclinic-env.yml *****
3 plays in workshop-answers/tp6/setup-petclinic-env.yml

PLAY [setup petclinic database] *****
skipping: no hosts matched

PLAY [setup petclinic application] *****
skipping: no hosts matched

PLAY [setup petclinic load balancer] *****

TASK [Gathering Facts] *****
task path: /vagrant/workshop-answers/tp6/setup-petclinic-env.yml:17
ok: [lbdev01]
META: ran handlers

TASK [platform/nginx : Install Nginx] *****
changed: [lbdev01]

TASK [platform/nginx : Define Nginx as a Service] *****
ok: [lbdev01]
META: role_complete for lbdev01
```

```
TASK [petclinic/lb : Produce petclinic nginx conf to nginx sites-available] *****
NOTIFIED HANDLER petclinic/lb : on__petclinic_lb__configuration_changed for lbdev01
changed: [lbdev01]

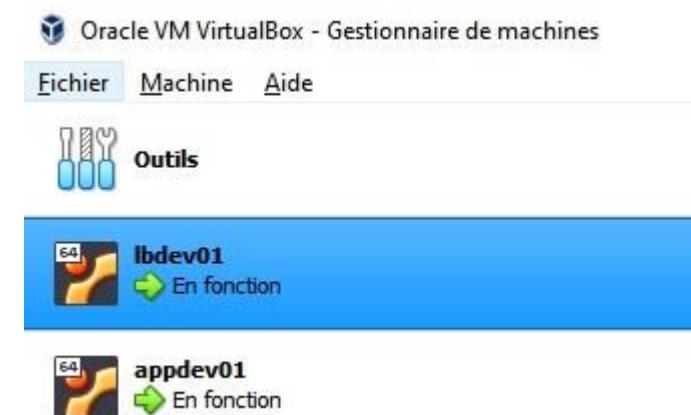
TASK [petclinic/lb : Validate nginx configuration] *****
changed: [lbdev01]

TASK [petclinic/lb : Enable petclinic site to be served by nginx] *****
changed: [lbdev01]
META: role_complete for lbdev01

RUNNING HANDLER [petclinic/lb : on__petclinic_lb__configuration_changed] *****
changed: [lbdev01]
META: ran handlers

PLAY RECAP *****
lbdev01 : ok=7    changed=5    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

==> lbdev01: Machine 'lbdev01' has a post `vagrant up` message. This is a message
==> lbdev01: from the creator of the Vagrantfile, and not from Vagrant itself:
==> lbdev01:
==> lbdev01: *** VM lbdev01 ready ***
```



Vous retrouvez ces VM au sein de VirtualBox



DEMO AVEC VAGRANT SOUS WINDOWS (3/3)



Code : login sur la VM lbdev01

```
$ vagrant ssh lbdev01
```

```
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-131-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Fri Nov  4 13:44:17 UTC 2022

System load: 0.0          Processes:           124
Usage of /: 5.5% of 38.70GB Users logged in:      0
Memory usage: 18%          IPv4 address for enp0s3: 10.0.2.15
Swap usage:  0%          IPv4 address for enp0s8: 172.28.128.11

0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
```



Code : une fois connecté sur lbdev01, ping sur appdev01 (une fois la VM provisionnée avec Vagrant)

```
$ ping 172.28.128.21
```

```
vagrant@lbdev01:~$ ping 172.28.128.21
PING 172.28.128.21 (172.28.128.21) 56(84) bytes of data.
64 bytes from 172.28.128.21: icmp_seq=1 ttl=64 time=1.45 ms
64 bytes from 172.28.128.21: icmp_seq=2 ttl=64 time=1.00 ms
^C
--- 172.28.128.21 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 1.003/1.226/1.450/0.223 ms
```



Code : rejouer le script ansible sur la VM lbdev01

```
$ vagrant provision lbdev01
```

```
...
PLAY RECAP ****
lbdev01 : ok=7    changed=3    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```



DEBUGGUER MES SCRIPTS SE POSER LES BONNES QUESTIONS

- Lancer le cycle
- Gestion de la précédence des variables
- Activer le verbose ansible (-vvv)
- Erreur dans les templates ?
- Inventaire correct ?
- ...



BONNES PRATIQUES & OPTIMISATION

BONNES PRATIQUES (1/2)

- Nommer vos tâches
- Utiliser des espaces et non des tabulations
- : **suivi d'un espace** est considéré comme un nouvel élément d'un dictionnaire. Protégez ce contenu avec des simples ou double quotes
- Double quoter les caractères réservés **[] {} > | * & ! % # ' @ ,**
- Ajuster le niveau de verbosité de vos messages avec le paramètre **verbosity**



Code : gestion de la verbosité d'un message

```
- name: message uniquement avec --vv et +
debug:
msg: "This only displays with ansible-playbook -vv+"
verbosity: 2
```

- Vérifiez la syntaxe de vos scripts avec l'option **--synthax-check**



Code :

```
$ ansible-playbook --syntax-check script.yml
```



BONNES PRATIQUES & OPTIMISATION

BONNES PRATIQUES (2/2)

- Commencez par écrire les tâches dans un playbook avant de les transformer en rôle
- Utilisez group_vars et hosts_vars plutôt que de surcharger l'inventaire de variables
- **Pensez 'état à atteindre' et non pas 'actions à appliquer'**
- Le playbook doit être **lisible** pour un technique (dev et ops)
- La plupart des réponses sont dans la doc (RTFM ☺)
- Ne pas avoir peur de développer un plugin/module pour gérer un problème complexe, dans ce cas respecter le principe d'idempotence
- Utilisez des **règles de codages**, exemple:
<https://github.com/apigee/edx-configuration/wiki/Ansible-Coding-Conventions>



BONNES PRATIQUES & OPTIMISATION

OPTIMISATION (1/3)

- Activez les **callback** timer, profile_tasks & profile_role dans le fichier ansible.cfg pour identifier les points de contention.



Code : activation des callbacks sous ansible.cfg

```
[defaults]
...
callbacks_enabled = timer, profile_tasks, profile_roles
```



```
$ ansible-playbook ...
...
=====
deploy-web-server : Install httpd and firewalld ----- 5.42s
deploy-web-server : Git checkout ----- 3.40s
Gathering Facts ----- 1.60s
deploy-web-server : Enable and Run Firewalld ----- 0.82s
deploy-web-server : firewalld permitt httpd service --- 0.72s
deploy-web-server : httpd enabled and running ----- 0.55s
deploy-web-server : Set Hostname on Site ----- 0.54s
deploy-web-server : Delete content & directory ----- 0.52s
deploy-web-server : Create directory ----- 0.41s
Deploy Web service ----- 0.04s
0:00:14.099
=====
deploy-web-server ----- 12.40s
gather_facts ----- 1.60s
include_role ----- 0.04s
~~~
total ----- 14.04s
```



Code : module setup avec subset gathering

```
- hosts: all
gather_facts: no
pre_tasks:
  - name: Collecte uniquement des faits network
    setup:
      gather_subset:
        - '!all'
        - '!any'
        - 'network'
tasks:
  - ...
```



BONNES PRATIQUES & OPTIMISATION

OPTIMISATION (2/3)

- Configurez le **parallélisme** avec le paramètre fork



Code : tuning du fork sous ansible.cfg

```
[defaults]
...
forks=20
```



Code : tuning du fork en ligne de commande

```
$ ansible-playbook ... --forks 20
```



Paramètre à ajuster en fonction de votre host ansible

- Pensez aux **tâches asynchrones** pour les opérations longues (backup,)



Code : tache asynchrone

```
tasks:
- name: une tache très longue
  ...
  async: 120 # temps maximum en secondes
  poll: 5    # interval de polling en secondes
```



A utiliser par exemple pour attendre le redémarrage d'un applicatif avant d'effectuer de nouvelles configurations

- Privilégiiez les **loop inline** des modules lorsque ceux-ci les supportent



Code : loop inline au sein d'un module

```
- name: Installation de packages
apt:
  pkg:
    - package1
    - package2
```

Inutile de gérer explicitement une loop



BONNES PRATIQUES & OPTIMISATION

OPTIMISATION (3/3)

■ Tunez votre connexion ssh

- Configurez la durée de vie de votre pool de connexion



Code : tuning de la connexion ssh sous ansible.cfg

```
[ssh_connection]  
ssh_args = -o ControlPersist=60s
```

une connexion idle sera conservée 60s

- Activez le pipelining ssh



Code : pipelining sous ansible.cfg

```
[ssh_connection]  
pipelining = True
```



Activer le **pipelining ssh** provoque un **conflit avec become**. A utiliser avec précaution.

BONNES PRATIQUES & OPTIMISATION

QUELQUES COMMANDES

- ansible, ansible-playbook, ansible-doc, ansible-vault, ansible-galaxy
- ansible-pull: -U repository
 - Idem ansible-playbook
 - Clone un dépôt contenant le playbook avant de l'exécuter
- ansible-console
 - Affiche un prompt permettant de lancer des commandes sur les hôtes d'un inventaire
- ansible-config
 - Gestion du fichier de configuration d'ansible
- ansible-inventory
 - Affiche un inventaire tel qu'ansible le voit

