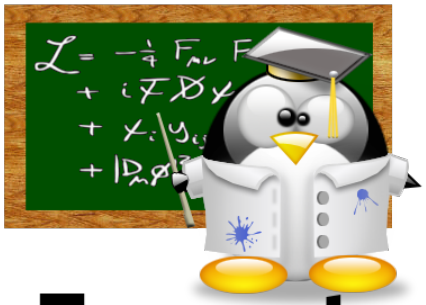


FORMATUX - Support de cours GNU/Linux

Automatisation - DevOPS

1.2, 23-08-2017

Table des matières

Préface

GNU/Linux est un **système d'exploitation** libre fonctionnant sur la base d'un **noyau Linux**, également appelé **kernel Linux**.

Linux est une implémentation libre du système **UNIX** et respecte les spécifications **POSIX**.

GNU/Linux est généralement distribué dans un ensemble cohérent de logiciels, assemblés autour du noyau Linux et prêt à être installé. Cet ensemble porte le nom de "**Distribution**".

- La plus ancienne des distributions est la distribution **Slackware**.
- Les plus connues et utilisées sont les distributions **Debian**, **RedHat** et **Arch**, et servent de base pour d'autres distributions comme **Ubuntu**, **CentOS**, **Fedora**, **Mageia** ou **Manjaro**.

Chaque distribution présente des particularités et peut être développée pour répondre à des besoins très précis :

- services d'infrastructure ;
- pare-feu ;
- serveur multimédia ;
- serveur de stockage ;
- etc.

La distribution présentée dans ces pages est la CentOS, qui est le pendant gratuit de la distribution RedHat. La distribution CentOS est particulièrement adaptée pour un usage sur des serveurs d'entreprises.

Crédits

Ce support de cours a été rédigé par les formateurs :

- Patrick Finet ;
- Antoine Le Morvan ;
- Xavier Sauvignon ;
- Nicolas Kovacs.

Licence

respect des droits de l'auteur.

BY : Paternité. Vous devez citer le nom de l'auteur original.

SA : Partage des Conditions Initiales à l'Identique.

- Licence Creative Commons-BY-SA : https://creativecommons.org/licenses/by-sa/3.0/fr/
- Licence Art Libre : http://artlibre.org/

Les documents de Formatux et leurs sources sont librement téléchargeables sur framagit :

- https://framagit.org/alemorvan/formatux.fr-support/

Vous y trouverez la dernière version de ce document.

A partir des sources, vous pouvez générer votre support de formation personnalisé. Nous vous recommandons le logiciel AsciidocFX téléchargeable ici : http://asciidocfx.com/

Gestion des versions

Table 1. Historique des versions du document

Version	Date	Observations
1.0	Avril 2017	Version initiale.
1.1	Juillet 2017	Ajout de généralités.
1.2	Aout 2017	Ajout du cours Jenkins et Rundeck
1.3	Février 2019	Ajout des cours Ansible Niveau 2, Ansistrano, Asciidoc, Terraform

Généralités devops

Attendus du cours

devops, automatisation, gestion de configuration, intégration continue, DSL.

Connaissances : ⚙⚙

Niveau technique : ★

Temps de lecture : 5 minutes

Le mouvement **devops** cherche à optimiser le travail de toutes les équipes intervenant sur un système d'information.

- Les développeurs (les **dev**) cherchent à ce que leurs applications soient déployées le plus souvent et le plus rapidement possible.
- Les administrateurs systèmes, réseaux ou de bases de données (les **ops**) cherchent à garantir la stabilité, la sécurité de leurs systèmes et leur disponibilité.

Les objectifs des **dev** et des **ops** sont donc bien souvent opposés, la communication entre les équipes parfois difficile : les dev et les ops n'utilisent pas toujours les mêmes éléments de langage.

- Il n'y a rien de plus frustrant pour un développeur que de devoir attendre la disponibilité d'un administrateur système pour voir la nouvelle fonctionnalité de son application être mise en ligne ;
- Quoi de plus frustrant pour un administrateur système de devoir déployer une nouvelle mise à jour applicative manuellement alors qu'il vient de finir la précédente ?

La philosophie devops regroupe l'ensemble des outils des deux mondes, offre un langage commun, afin de faciliter le travail des équipes avec comme objectif la performance économique pour l'entreprise.

Le travail des développeurs et des administrateurs doit être simplifié afin d'être automatisé avec des outils spécifiques.

Le vocabulaire DEVOPS

- le **build** : concerne la conception de l'application ;
- le **run** : concerne la maintenance de l'application ;
- le **change** : concerne l'évolution de l'application.
- l'**intégration continue** (Continuous Integration CI) : chaque modification d'un code source entraîne une vérification de non-régression de l'application.

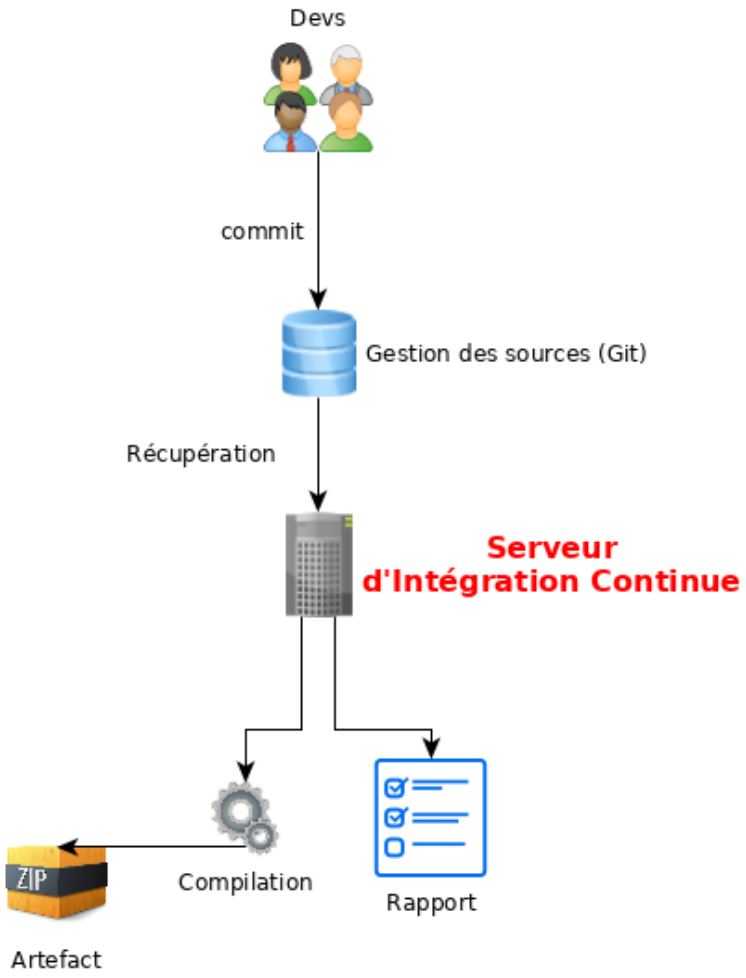


Figure 1. Schéma de fonctionnement de l'intégration continue

- **automation** (automatisation) : fonctionnement d'un système sans intervention humaine, automatisation d'une suite d'opération.
- **idempotence** : une opération est idempotente si elle a le même effet qu'on l'applique une ou plusieurs fois. Les outils de gestion de configuration sont généralement idempotent.
- **orchestration** : processus automatique de gestion d'un système informatique.
- **provisionning** : processus d'allocation automatique des ressources.

Pourquoi scripter en Bash n'est pas considéré comme de l'automatisation ?

Ou les langages impératifs contre les languages déclaratifs...

Même si la connaissance du **Bash** est une exigence de base pour un administrateur système, celui-ci est un langage de programmation interprété "**impératif**". Il exécute les instructions les unes à la suite des autres.

Les langages dédiés au domaine (**DSL** Domain Specific Language) comme Ansible ou Puppet, quant à eux, ne spécifient pas les étapes à réaliser mais l'état à obtenir.

Parce qu'Ansible ou Puppet utilisent un langage déclaratif, ils sont très simples. Il suffit de leur dire "Fais cette action" ou "Met le serveur dans cet état". Ils considéreront l'état désiré indépendamment de l'état initial ou du contexte. Peu importe dans quel état le serveur se situe au départ, les étapes à franchir pour arriver au résultat, le serveur est mis dans l'état désiré avec un rapport de succès (avec ou sans changement d'état) ou d'échec.

La même tâche d'automatisation en bash nécessiterait de vérifier tous les états possibles, les autorisations, etc. afin de déterminer la tâche à accomplir avant de lancer l'exécution de la commande, souvent en imbriquant de nombreux "si", ce qui complique la tâche, l'écriture et la maintenance du script.

Les outils devops

- Outils de gestion de configuration
  - Puppet
  - Ansible
  - Saltstack
  - Chef
- Intégration continue
  - Jenkins
  - Gitlab-ci
- Orchestration des tâches
  - Rundeck

Gestion de configurations avec Puppet

Il est difficile de s'appuyer sur des processus manuels ou des scripts personnalisés pour accomplir des tâches répétitives.

Lorsque les environnements grossissent ou que les équipes accueillent de plus en plus de techniciens, ces méthodes sont difficiles à maintenir et à optimiser. Elles peuvent causer des risques pour la sécurité du système, incluant des erreurs de configuration, ce qui au final, peut faire réduire la productivité.

La gestion automatique des configurations élimine beaucoup de travail manuel et augmente la réactivité des équipes. Cette réactivité devient de plus en plus importante avec la montée en puissance de la virtualisation, qui révolutionne notre gestion du cycle de vie des serveurs : durée de vie plus courte, déploiement plus rapide, configurations plus standardisées et conformes.

Parmi les systèmes de gestion de configurations, plusieurs systèmes ont fait leur apparition :

- puppet ;
- chef ;
- ansible ;
- etc.

Des outils ont également fait leur apparition pour encore faciliter l'administration de ces systèmes :

- geppetto : un environnement de développement (IDE) pour puppet ;
- the foreman : un gestionnaire de cycle de vie complet des serveurs.

La gestion de configuration

La gestion de configuration est le processus de standardisation des configurations de ressources et l'assurance de leur état dans l'infrastructure informatique, avec des méthodes automatisées mais agiles. Le management de configurations est devenu critique pour le succès des projets informatiques.

Concrètement, lors de l'ajout d'un nouveau serveur au sein d'une infrastructure informatique complexe, les administrateurs système ne doivent pas perdre de temps pour la configuration des briques de base du système : la configuration des services NTP, DNS, SMTP, SSH, la création des comptes utilisateurs, etc… doit être totalement automatisée et transparente aux équipes.

L'utilisation d'un gestionnaire de configuration doit également permettre d'installer un clone de serveur d'une manière totalement automatisée, ce qui peut être pratique pour des environnements multiples (Développement → Intégration → Pré-production → Production).

La combinaison d'outils de gestion de configuration avec l'utilisation de dépôts de gestion de versions, comme « git », permet de conserver un historique des modifications apportées au

système.

Puppet

Puppet a été conçu pour fonctionner en mode client-serveur. Son utilisation en mode de fonctionnement « autonome » est également possible et facile. La migration vers un système de clients « Puppet / Master Puppet » n'est pas d'une réalisation complexe.

Puppet est un logiciel d'automatisation qui rend simple pour l'administrateur système le provisionnement (la description matérielle d'une machine virtuelle), la configuration et la gestion de l'infrastructure tout au long de son cycle de vie. Il permet de décrire l'état de configuration d'un ensemble hétérogène de stations de travail ou de serveurs et de s'assurer que l'état réel des machines correspond bien à l'état demandé.

Par sa structure de langage, il fait le lien entre les bonnes pratiques, le cahier de procédures et l'état effectif des machines.

Vocabulaire Puppet

- **Noeud** (Node) : serveur ou poste de travail administré par Puppet ;
- **Site** : ensemble des noeuds gérés par le Puppet Master ;
- **Classe** : moyen dans Puppet de séparer des morceaux de code ;
- **Module** : unité de code Puppet qui est réutilisable et pouvant être partagé ;
- **Catalogue** : ensemble des classes de configuration à appliquer à un nœud ;
- **Facter** : librairie multi-plateforme qui fournit à Puppet sous forme de variables les informations propres au système (nom, adresse ip, système d'exploitation, etc.) ;
- **Ressource** (Resource): objet que Puppet peut manipuler (fichier, utilisateur, service, package, etc.) ;
- **Manifeste** (Manifest) : regroupe un ensemble de ressource.

Architecture

Puppet conseille de coupler son fonctionnement avec un gestionnaire de version type « git ».

Un serveur PuppetMaster contient la configuration commune et les points de différence entre machines ;

Chaque client fait fonctionner puppetd qui :

- applique la configuration initiale pour le nœud concerné ;
- applique les nouveautés de configuration au fil du temps ;
- s'assure de manière régulière que la machine correspond bien à la configuration voulue.

La communication est assurée via des canaux chiffrés, en utilisant le protocole https et donc TLS

(une mini-pki est fournie).

Toute la configuration (le référentiel) de Puppet est centralisée dans l'arborescence /etc/puppet du serveur de référence :

- **/etc/puppet/manifests/site.pp** : est le premier fichier analysé par Puppet pour définir son référentiel. Il permet de définir les variables globales et d'importer des modules ;
- **/etc/puppet/manifests/node.pp** : permet de définir les nœuds du réseau. Un nœud doit être défini par le nom FQDN de la machine ;
- **/etc/puppet/modules/<module>** : sous-répertoire contenant un module.

Installation

Les dépôts Puppets doivent être activés :

```
[root]# vim /etc/yum/yum.repos.d/puppetlabs.repo
[puppetlabs-products]
name=Puppet Labs Products El 6 - $basearch
baseurl=http://yum.puppetlabs.com/el/6/products/$basearch
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=1
gpgcheck=1

[puppetlabs-deps]
name=Puppet Labs Dependencies EL 6 - $basearch
baseurl=http://yum.puppetlabs.com/el/6/dependencies/$basearch
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=1
gpgcheck=1
```

puis :

```
[root]# yum update
[root]# yum install puppet
```

Hello world

Pour fonctionner, le client autonome puppet a besoin d'un fichier appelé manifeste, dont l'extension sera en « .pp ».

Le langage utilisé est le ruby.

Créer un fichier /root/config-init.pp :

```
[root]# vim /root/config-init.pp
file {'helloworld':
  path => '/tmp/helloworld',
  ensure => present,
  mode => 0640,
  content => "Helloworld via puppet ! "
}
```

Exécuter ce manifeste avec la commande puppet :

```
[root]# puppet apply /root/config-init.pp
Notice : Compiled catalog for centos6 in environnement production in 0.31 seconds
Notice: /Stage[main]/main/File[helloworld]/ensure: created
Notice: Finished catalog run in 0.07 seconds
```

Les modules Puppet

Les modules complémentaires à Puppet peuvent être téléchargés sur le site puppetlabs.

L'installation d'un module complémentaire pourra se faire, soit directement depuis internet, soit par le téléchargement manuel de l'archive .tar.gz.

Depuis internet :

```
puppet module install nomdumodule
```

Une commande de recherche de modules existe :

```
puppet module search nomdumodule
```

Pour rechercher les modules installés :

```
puppet module list
```

Et pour les mettre à jour :

```
puppet module upgrade nomdumodule
```

Pensez à préciser le proxy dans la commande puppet en exportant les variables http_proxy et https_proxy :

```
export http_proxy=http://10.10.10.7:8080
export https_proxy=http://10.10.10.7:8080
```

Sans connexion internet

Sans connexion internet, un module peut être installé en fournissant dans la commande puppet le chemin vers le fichier tar.gz :

```
puppet module install ~/nomdumodule.tar.gz --ignore-dependencies
```

Grâce aux modules du dépôt Puppet, les possibilités de l'outil sont quasiment infinies. Le téléchargement et l'utilisation des modules du dépôt permettent un gain de temps confortable car l'outil nécessite peu de compétences en développement.

Documentation

La liste des types et leurs attributs est consultable en ligne :

- https://docs.puppetlabs.com/references/latest/type.html

Une documentation de formation est également consultable :

- https://doc.puppetlabs.com/learning/introduction.html

Commandes utiles

Lister les objets connus par puppet :

```
puppet describe -l
```

Lister les valeurs possibles d'une ressource :

```
puppet describe user
```

Lister les objets du système :

```
puppet resource user
```

Cas concrets

La création d'un noeud (node)

Le code du manifeste doit être découpé en classe pour des raisons de maintenance et d'évolutivité.

Un objet de type node recevra les classes à exécuter. Le node « default » est automatiquement exécuté.

Le fichier site.pp contiendra l'ensemble du code :

```
node default {
  include init_services
}
```

Gestion des services

La classe init_services contient les services qui doivent être lancés sur le nœud et ceux qui doivent être stoppés :

```
class init_services {

  service { ["sshd","NetworkManager","iptables","postfix","puppet","rsyslog","sssd",
"vmware-tools"]:
    ensure => 'running',
    enable => 'true',
  }

  service { ["atd","cups","bluetooth","ip6tables","ntpd","ntpdate","snmpd","snmptradp"]:
    ensure => 'stopped',
    enable => 'false',
  }
}
```

La ligne ensure ⇒ a pour effet de démarrer ou non un service, tandis que la ligne enable ⇒ activera ou non le démarrage du service au démarrage du serveur.

Gestion des utilisateurs

La classe create_users contiendra les utilisateurs de notre système. Pensez à ajouter l'appel de cette classe dans le node default !

```
class create_users {
  user { 'antoine':
    ensure => present,
    uid => '5000',
    gid => 'users',
    shell => '/bin/bash',
    home => '/home/antoine',
    managehome => true,
  }
}
```

```
node default {
  include init_services
  include create_users
}
```

Au départ du personnel pour mutation, il sera aisé de supprimer son compte en remplaçant la ligne ensure ⇒ present par ensure ⇒ absent et en supprimant le reste des options.

La directive managehome permet de créer les répertoires personnels à la création des comptes.

La création des groupes est toute aussi aisée :

```
group { "DSI":
  ensure => present,
  gid => 1001
}
```

Gestion des dossiers et des fichiers

Un dossier peut être créé avec la ressource « file » :

```
file { '/etc/skel/boulot':
  ensure => directory,
  mode   => 0644,
}
```

Un fichier peut être copié d'un répertoire vers un autre :

```
file { '/STAGE/utilisateurs/gshadow':
  mode   => 440,
  owner  => root,
  group  => root,
  source => "/etc/gshadow"
}
```

Modification de valeurs

La commande augeas, développée par la société RedHat, permet de modifier les valeurs des variables dans les fichiers de configuration. Son utilisation peut s'avérer autant puissante que complexe.

Un usage minimaliste serait par exemple :

```
augeas { "Modification default login defs" :
    context => "/files/etc/login.defs",
    changes => ["set UID_MIN 2000","set GID_MIN 700","set PASS_MAX_DAYS 60"],
}
```

Le contexte est suffixé de « /files/ » pour préciser qu'il s'agit d'un système de fichiers local.

Exécution d'une commande externe

La commande exec est utilisée pour lancer une commande externe :

```
exec { "Redirection":
  command => "/usr/sbin/useradd -D > /STAGE/utilisateurs/defaut",
}
```

De manière générale, les commandes et les fichiers doivent être décrits en absolu dans les manifestes.

Rappel : la commande whereis fournit le chemin absolu d'une commande.

Ansible

🎓 Objectifs

☐ Mettre en oeuvre Ansible ;
☐ Appliquer des changements de configuration sur un serveur ;
☐ Créer des playbooks Ansible.

Ansible centralise et automatise les tâches d'administration. Il est :

- sans **agent** (il ne nécessite pas de déploiements spécifiques sur les clients),
- **idempotent** (effet identique à chaque exécution)
- et utilise le protocole **SSH** pour configurer à distance les clients Linux.

L'interface graphique web Ansible Tower est payante.

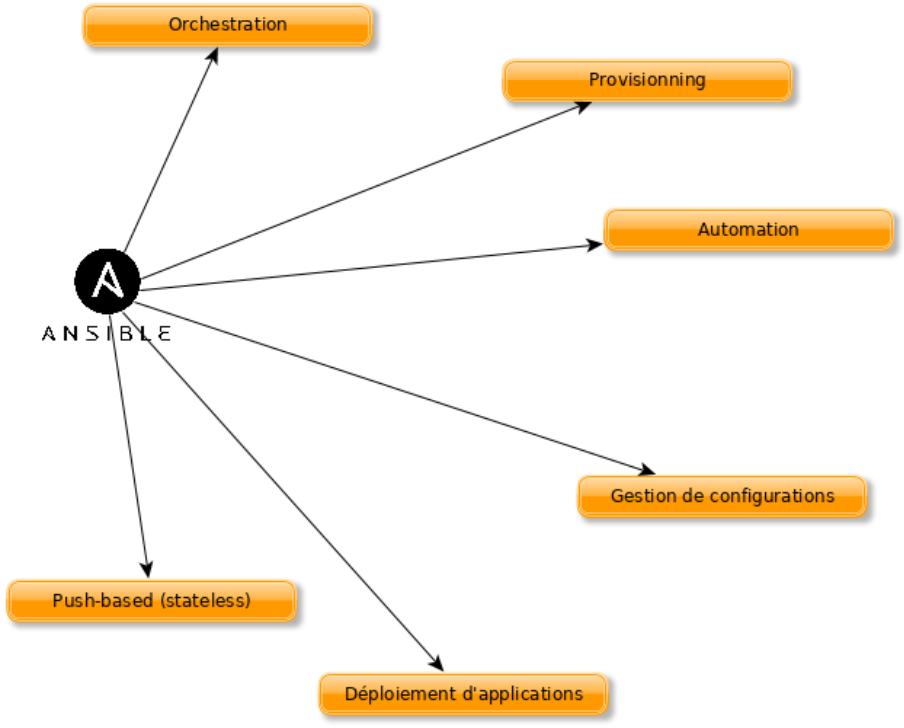


Figure 2. Les fonctionnalités d'Ansible

L'ouverture des flux SSH vers l'ensemble des clients depuis le serveur Ansible font de lui un elément critique de l'architecture qu'il faudra attentivement surveiller.

Le vocabulaire ansible

- Le **poste de gestion** : la machine sur laquelle Ansible est installée. Ansible étant **agentless**, aucun logiciel n'est déployé sur les serveurs gérés.
- L'**inventaire** : un fichier contenant les informations concernant les serveurs gérés.
- Les **tâches** (tasks) : une tâche est un bloc définissant une procédure à exectuer (par exemple créer un utilisateur ou un groupe, installer un paquet logiciel, etc.).
- Un **module** : un module rend abstrait une tâche. Il existe de nombreux modules fournis par Ansible.
- Les **playbooks** : un fichier simple au format yaml définissant les serveurs cibles et les tâches devant être effectuées.
- Un **rôle** : un rôle permet d'organiser les playbooks et tous les autres fichiers nécessaires (modèles, scripts, etc.) pour faciliter le partage et la réutilisation du code.
- Les **facts** : ce sont des variables globales contenant des informations à propos du système (nom de la machine, version du système, interface et configuration réseau, etc.).
- les **handlers**: il sont utilisés pour provoquer un arrêt ou un redémarrage d'un service en cas de changement.

Installation sur le serveur de gestion

Ansible est disponible dans le dépôt *EPEL* :

- Installation d'EPEL :

```
$ sudo yum install epel-release
```

La configuration du serveur se situe sous `/etc/ansible`.

Deux fichiers de configuration :

- Le fichier de configuration principal `ansible.cfg` : commandes, modules, plugins, configuration ssh ;
- Le fichier inventaire de gestion des machines clientes `hosts` : déclaration des clients, des groupes.

Le fichier /etc/ansible/hosts

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers.

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

...
```

Par exemple, un groupe `centos7` est créé en insérant le bloc suivant dans ce fichier :

Création d'un groupe d'hôtes dans /etc/ansible/hosts

```
[centos7]
172.16.1.217
172.16.1.192
```

Utilisation en ligne de commande

La commande `ansible` lance une tâche sur un ou plusieurs hôtes cibles.

Syntaxe de la commande ansible

```
ansible <host-pattern> [-m module_name] [-a args] [options]
```

Exemples :

- Lister les hôtes appartenant à un groupe :

```
ansible {{group}} --list-hosts
```

- Pinger un groupe d'hôtes avec le module `ping` :

```
ansible {{group}} -m ping
```

- Afficher des facts d'un groupe d'hôtes avec le module `setup` :

```
ansible {{group}} -m setup
```

- Exécuter une commande sur un groupe d'hôtes en invoquant le module `command` avec des arguments :

```
ansible {{group}} -m command -a '{{commande}}'
```

- Exécuter une commande avec les privilèges d'administrateur :

```
ansible {{group}} --become --ask-become-pass -m command -a '{{commande}}'
```

- Exécuter une commande en utilisant un fichier d'inventaire personnalisé :

```
ansible {{group}} -i {{inventory_file}} -m command -a '{{commande}}'
```

Table 2. Options principales de la commande ansible

Option	Information
`-a 'arguments'`	Les arguments à passer au module.
`-b -K`	Demande un mot de passe et lance la commande avec des privilèges supérieurs.
`--user=utilisateur`	Utilise cet utilisateur pour se connecter à l'hôte cible au lieu d'utiliser l'utilisateur courant.
`--become -user=utilisateur`	Exécute l'opération en tant que cet utilisateur (défaut : `root`).
`-C`	Simulation. Ne fait pas de changement sur la cible mais la teste pour voir ce qui devrait être changé.
`-m module`	Exécute le module appelé

Tester avec le module ping

Par défaut la connexion par mot de passe n'est pas autorisée par Ansible.

Décommenter la ligne suivante de la section `[defaults]` dans le fichier de configuration `/etc/ansible/ansible.cfg` :

```
ask_pass      = True
```

Lancer un `ping` sur chacun des serveurs du groupe CentOS 7 :

```
# ansible centos7 -m ping
SSH password:
172.16.1.192 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
172.16.1.217 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Le mot de passe `root` des serveurs distants vous est demandé, ce qui pose un problème de sécurité…

Authentification par clef

L'authentification par mot de passe va être remplacée par une authentification par clefs privée/publique beaucoup plus sécurisée.

Création d'une clef SSH

La bi-clefs va être générée avec la commande `ssh-keygen` :

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:RpYuJzkkaeZzve8La8Xd/8kTTE8t43DeS+L7WB26WF8 root@ansible-srv
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|     .   .       |
|    = . +       .|
|   + o *    . +.o|
|    o * S. . *o*.|
|     o * .o ..=+=|
|        o.  .oooE|
|        .+  o.*o+|
|       ...+o +o=+|
+----[SHA256]-----+
```

La clef publique peut être copiée sur les serveurs :

```
# ssh-copy-id root@172.16.1.192
# ssh-copy-id root@172.16.1.217
```

Re-commenter la ligne suivante de la section `[defaults]` dans le fichier de configuration `/etc/ansible/ansible.cfg` pour empêcher l'authentification par mot de passe :

```
#ask_pass      = True
```

Test d'authentification par clef privée

Pour le prochain test, le module `shell`, permettant l'exécution de commandes à distance, est utilisé :

```
# ansible centos7 -m shell -a "uptime"
172.16.1.192 | SUCCESS | rc=0 >>
 12:36:18 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00

172.16.1.217 | SUCCESS | rc=0 >>
 12:37:07 up 57 min,  1 user,  load average: 0.00, 0.00, 0.00
```

Aucun mot de passe n'est demandé, l'authentification par clé privée/publique fonctionne !

Exemple de connexion à une instance Cloud Amazon ECS

Lors de la création d'une instance Amazon, une clef privée est créée et téléchargée sur le poste local.

Ajout de la clef dans l'agent SSH :

```
ssh-add path/to/fichier.pem
```

Lancement des `facts` sur les serveurs aws :

```
ansible aws --user=ec2-user --become -m setup
```

Pour une image ubuntu, il faudra utiliser l'utilisateur ubuntu :

```
ansible aws --user=ubuntu --become -m setup
```

Utilisation

Ansible peut être utilisé depuis l'interpréteur de commandes ou via des playbooks.

Les modules

La liste des modules classés par catégories se trouve à l'adresse http://docs.ansible.com/ansible/modules_by_category.html. Ansible en propose plus de 750 !

Un module s'invoque avec l'option `-m` de la commande ansible.

Il existe un module pour chaque besoin ou presque ! Il est donc conseillé, au lui d'utiliser le module shell, de chercher un module adapté au besoin.

Chaque catégorie de besoin dispose de son module. En voici une liste non exhaustive :

Table 3. Catégories de modules

Type	Exemples
Gestion du système	`user` (création des utilisateurs), `group` (gestion des groupes), etc.
Gestion des logiciels	`yum`, `apt`, `pip`, `npm`
Gestion des fichiers	`copy`, `fetch`, `lineinfile`, `template`, `archive`
Gestion des bases de données	`mysql`, `postgresql`, `redis`

Type	Exemples
Gestion du cloud	`amazon S3`, `cloudstack`, `openstack`
Gestion d'un cluster	`consul`, `zookeeper`
Envoyer des commandes	`shell`, `script`, `expect`
Gestion des messages	
Gestion du monitoring	
Gestion du réseau	`get_url`
Gestion des notifications	
Gestion des sources	`git`, `gitlab`

Exemple d'installation logiciel

Le module `yum` permet d'installer des logiciels sur les clients cibles :

```
# ansible centos7 -m yum -a name="httpd"
172.16.1.192 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
      \n\nComplete!\n"
    ]
}
172.16.1.217 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
      ...
    \n\nComplete!\n"
    ]
}
```

Le logiciel installé étant un service, il faut maintenant le démarrer avec le module `service` (CentOS 6) ou `systemd` (CentOS 7) :

```
# ansible centos7 -m systemd -a "name=httpd state=started"
172.16.1.192 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
172.16.1.217 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
```

Les playbooks

Les `playbooks` ansible décrivent une politique à appliquer à des systèmes distants, pour forcer leur configuration. Les playbooks sont écrits dans un format texte facilement compréhensible regroupant un ensemble de tâches : le format `yaml`.

En savoir plus sur le yaml : http://docs.ansible.com/ansible/YAMLSyntax.html

Syntaxe de la commande ansible-playbook

```
ansible-playbook <fichier.yml> ... [options]
```

Les options sont identiques à la commande ansible.

La commande renvoi les codes d'erreurs suivants :

Table 4. Codes de sortie de la commande ansible-playbook

Code	Erreur
`0`	OK ou aucun hôte correspondant
`1`	Erreur
`2`	Un ou plusieurs hôtes sont en échecs
`3`	Un ou plusieurs hôtes ne sont pas joignables
`4`	Erreur d'analyse
`5`	Mauvaises options ou options incomplètes
`99`	Execution interrompue par l'utilisateur
`250`	Erreur inattendue

Exemple de playbook Apache et MySQL

Le playbook suivant permet d'installer Apache et MySQL sur nos serveurs cibles :

```
---
- hosts: centos7
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd,php,php-mysqli state=latest
  - name: ensure httpd is started
    systemd: name=httpd state=started
  - name: ensure mysql is at the latest version
    yum: name=mysql-server state=latest
  - name: ensure mysqld is started
    systemd: name=mysqld state=started
```

L'exécution du playbook s'effectue avec la commande `ansible-playbook` :

```
$ ansible-playbook test

PLAY [centos7] *****************************************************************

TASK [setup] *******************************************************************
ok: [172.16.1.192]
ok: [172.16.1.217]

TASK [ensure apache is at the latest version] **********************************
ok: [172.16.1.192]
ok: [172.16.1.217]

TASK [ensure httpd is started] *************************************************
changed: [172.16.1.192]
changed: [172.16.1.217]

TASK [ensure mysql is at the latest version] ***********************************
changed: [172.16.1.192]
changed: [172.16.1.217]

TASK [ensure mysqld is started] ************************************************
changed: [172.16.1.192]
changed: [172.16.1.217]

PLAY RECAP *********************************************************************
172.16.1.192               : ok=5    changed=3    unreachable=0    failed=0
172.16.1.217               : ok=5    changed=3    unreachable=0    failed=0
```

Exemple de préparation d'un noeud MySQL

Dans ce playbook, un serveur va être installé et configuré pour héberger une base de données MySQL.

Le playbook utilise :

- Des variables ;
- Ajoute des lignes dans le fichier `/etc/hosts` ;
- Installe et démarre MariaDB ;
- Créer une base de données, un utilisateur et lui donne tous les droits sur les bases de données.

```
---

- hosts: centos7
  become: yes
  vars:
    mysqlpackage: "mariadb-server,MySQL-python"
    mysqlservice: "mariadb"
    mysql_port: "3306"
    dbuser: "synchro"
    dbname: "mabase"
    upassword: "M!rro!r"

  tasks:
    - name: configurer le noeud 1 dans /etc/hosts
      lineinfile:
              dest: /etc/hosts
              line: "13.59.197.48 mirroir1.local.lan mirroir1"
              state: present
    - name: configurer le noeud 2 dans /etc/hosts
      lineinfile:
              dest: /etc/hosts
              line: "52.14.125.109 mirroir2.local.lan mirroir2"
              state: present
    - name: mariadb installe et a jour
      yum: name="{{ mysqlpackage }}" state=latest
    - name: mariadb est demarre
      service: name="{{ mysqlservice }}" state=started
    - name: creer la base de donnee
      mysql_db: name="{{ dbname }}" state=present
    - name: creer un utilisateur
      mysql_user: name="{{ dbuser }}" password="{{ upassword }}" priv=*.*:ALL host='%'
state=present
    - name: restart mariadb
      service: name="{{ mysqlservice }}" state=restarted
...
```

La gestion des boucles

Il existe plusieurs type de boucles sous Ansible :

- `with_items`
- `with_file`
- `with_fileglob`
- ...

Exemple d'utilisation, création de 3 utilisateurs :

```
- name: ajouter des utilisateurs
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  with_items:
     - antoine
     - xavier
     - patrick
```

Les rôles

Un rôle Ansible est une unité favorisant la réutilisabilité des playbooks.

Un squelette de rôle, servant comme point de départ du développement d'un rôle personnalisé, peut être généré par la commande `ansible-galaxy` :

```
$ ansible-galaxy init formatux
```

La commande aura pour effet de générer l'arborescence suivante pour contenir le rôle `formatux` :

```
$ tree formatux
formatux/
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

La commande ansible-galaxy

La commande `ansible-galaxy` gère des rôles en utilisant le site galaxy.ansible.com.

Syntaxe de la commande ansible-galaxy

```
ansible-galaxy [import|init|install|login|remove|...]
```

Table 5. Sous-commandes de la commande ansible-galaxy

Sous-commandes	Observations
`install`	installe un rôle
`remove`	retire un ou plusieurs rôles
`init`	génère un squelette de nouveau rôle
`import`	importe un rôle depuis le site web galaxy. Nécessite un login au préalable.

Ansible Niveau 2

🎓 Objectifs

- ☐ Utiliser les variables ;
- ☐ Mettre en oeuvre des boucles et des conditions ;
- ☐ Gérer les fichiers ;
- ☐ Envoyer des notifications et réagir ;
- ☐ Gérer les fichiers ;
- ☐ Créer des tâches asynchrones.

Les variables

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_variables.html

Sous Ansible, il existe deux types de variables :

- la valeur simple
- le dictionnaire

Une variable peut être définie dans un playbook, dans un rôle ou depuis la ligne de commande.

Par exemple, depuis un playbook :

```
---
- hosts: apache1
  remote_user: root
  vars:
    port_http: 80
    service:
      debian: apache2
      centos: httpd
```

ou depuis la ligne de commande :

```
$ ansible-playbook deploy-http.yml --extra-vars "service=httpd"
```

Une fois définie, une variable peut être utilisée en l'appellant entre doubles accolades :

- {{ port_http }} pour la valeur simple
  - {{ service['centos'] }} ou {{ service.centos }} pour le dictionnaire.

Par exemple :

```
  tasks:
  - name: make sure apache is started
    service: name={{ service['centos'] }} state=started
```

Evidemment, il est également possible d'accéder aux variables globales d'Ansible (type d'OS, adresses IP, nom de la VM, etc.).

Externaliser les variables

Les variables peuvent être déportées dans un fichier externe au playbook, auquel cas il faudra définir ce fichier dans le playbook avec la directive `vars_files` :

```
---
- hosts: apache1
  remote_user: root
  vars_files:
    - mesvariables.yml
```

Le fichier mesvariables.yml

```
---
port_http: 80
service:
  debian: apache2
  centos: httpd
```

Afficher une variable

Pour afficher une variable, il faut activer le mode `debug` de la façon suivante :

Afficher une variable

```
- debug:
  msg: "Afficher la variable : {{ service['debian'] }}"
```

Enregistrer le retour d'une tâche

Pour enregistrer le retour d'une tâche et pouvoir y accéder plus tard, il faut utiliser le mot clef `register` à l'intérieur même de la tâche.

Utilisation d'une variable stockée

```
  tasks:
  - name: contenu de /home
    shell: ls /home
    register: homes

  - name: affiche le premier repertoire
    debug:
      var: homes.stdout_lines[0]

  - name: affiche le second repertoire
    debug:
      var: homes.stdout_lines[1]
```

Les chaînes de caractères composant la variable enregistrée sont accessibles via la valeur `stdout` (ce qui permet de faire des choses comme `homes.stdout.find("core") != -1`), de les exploiter en utilisant une boucle (voir `with_items`), ou tout simplement par leurs indices comme vu dans l'exemple précédent.

La gestion des boucles

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_loops.html

Il existe plusieurs types de boucles sous Ansible, en fonction de l'objet que vous voulez manipuler :

- `with_items`
- `with_file`
- `with_dict`
- `with_fileglob`
- ...

Exemple d'utilisation, création de 3 utilisateurs :

```
- name: ajouter des utilisateurs
  user:
    name: "{{ item }}"
    state: present
    groups: "users"
  with_items:
     - antoine
     - kevin
     - nicolas
```

Nous pouvons reprendre l'exemple vu durant l'étude des variables stockées pour l'améliorer :

Utilisation d'une variable stockée

```
  tasks:
  - name: contenu de /home
    shell: ls /home
    register: homes

  - name: affiche le nom des repertoires
    debug:
      msg: "Dossier => {{ item }}"
    with_items:
      - "{{ homes.stdout_lines }}"
```

Une fois un dictionnaire créé, celui-ci peut être parcouru en utilisant la variable `item` comme index :

```
---
- hosts: ansiblecli
  remote_user: ansible
  become: true
  vars:
    users:
      antoine:
        group: users
        state: present
      erwan:
        group: users
        state: absent

  tasks:

  - name: creer les utilisateurs
    user:
      name: "{{ item }}"
      group: "{{ users[item]['group'] }}"
      state: "{{ users[item]['state'] }}"
    with_items: "{{ users }}"
```

Les conditions

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_conditionals.html

L'instruction `when` est très pratique dans de nombreux cas : ne pas effectuer certaines actions sur certains type de serveur, si un fichier ou un n'utilisateur n'existe pas, etc.

Derrière l'instruction `when` les variables ne nécessitent pas de doubles accolades (il s'agit en fait d'expressions Jinja2...).

```
tasks:
  - name: "ne redemarre que les OS Debian"
    command: /sbin/shutdown -r now
    when: ansible_os_family == "Debian"
```

Les conditions peuvent être regroupées avec des parenthèses :

```
tasks:
  - name: "ne redémarre que les CentOS version 6 et Debian version 7"
    command: /sbin/shutdown -r now
    when: (ansible_distribution == "CentOS" and ansible_distribution_major_version ==
"6") or
          (ansible_distribution == "Debian" and ansible_distribution_major_version ==
"7")
```

Les conditions correspondant à un ET logique peuvent être fournies sous forme de liste :

```
tasks:
  - name: "ne redémarre que les CentOS 6"
    command: /sbin/shutdown -r now
    when:
      - ansible_distribution == "CentOS"
      - ansible_distribution_major_version == "6"
```

Le résultat de la variable peut aussi être utilisé conjointement aux conditions `when` et son absence de contenu évalué :

```
  tasks:

  - name: check if /data exists
    command: find / -maxdepth 1 -type d -name data
    register: datadir

  - name: print warning if /data does not exist
    debug: msg="Le dossier /data n'existe pas..."
    when: datadir.stdout == ""
```

La gestion des fichiers

Plus d'informations sur http://docs.ansible.com/ansible/latest/list_of_files_modules.html

En fonction de votre besoin, vous allez être amenés à utiliser différents modules Ansible pour modifier les fichiers de configuration du système.

Le module `ini_file`

Lorsqu'il s'agit de modifier un fichier de type INI (section entre [] puis paires de clef=valeur), le plus simple est d'utiliser le module ini_file.

Le module nécessite :

- La valeur de la section
- Le nom de l'option
- La nouvelle valeur

Exemple d'utilisation :

```
- name: change value on inifile
  ini_file: dest=/path/to/file.ini section=SECTIONNAME option=OPTIONNAME
value=NEWVALUE
```

Le module `lineinfile`

Pour s'assurer qu'une ligne est présente dans un fichier, ou lorsqu'une seule ligne d'un fichier doit être ajoutée ou modifiée.

Voir http://docs.ansible.com/ansible/latest/lineinfile_module.html.

Dans ce cas, la ligne à modifier d'un fichier sera retrouvée à l'aide d'une regexp.

Par exemple, pour s'assurer que la ligne commençant par 'SELINUX=' dans le fichier /etc/selinux/config contiennent la valeur `enforcing` :

```
- lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'
```

Le module `copy`

Lorsqu'un fichier doit être copié depuis le serveur Ansible vers un ou plusieurs hosts, dans ce cas il est préférable d'utiliser le module `copy` :

```
- copy:
    src: /data/ansible/sources/monfichier.conf
    dest: /etc/monfichier.conf
    owner: root
    group: root
    mode: 0644
```

Le module `fetch`

Lorsqu'un fichier doit être copié depuis un serveur distant vers le serveur local.

Ce module fait l'inverse du module `copy`.

```
- fetch:
    src: /etc/monfichier.conf
    dest: /data/ansible/backup/monfichier-{{ inventory_hostname }}.conf
    flat: yes
```

Le module `template`

Ansible et son module `template` utilisent le système de template Jinja2 (http://jinja.pocoo.org/docs/) pour générer des fichiers sur les hôtes cibles.

```
- template:
    src: /data/ansible/templates/monfichier.j2
    dest: /etc/monfichier.conf
    owner: root
    group: root
    mode: 0644
```

Il est possible d'ajouter une étape de validation si le service ciblé le permet (par exemple apache avec la commande `apachectl -t`) :

```
- template:
    src: /data/ansible/templates/vhost.j2
    dest: /etc/httpd/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
    validate: '/usr/sbin/apachectl -t'
```

Le module `get_url`

Pour télécharger vers un ou plusieurs hôtes des fichiers depuis un site web ou ftp.

```
- get_url:
    url: http://site.com/archive.zip
    dest: /tmp/archive.zip
    mode: 0640
    checksum: sha256:f772bd36185515581aa9a2e4b38fb97940ff28764900ba708e68286121770e9a
```

En fournissant un checksum du fichier, ce dernier ne sera pas re-téléchargé s'il est déjà présent à l'emplacement de destination et que son checksum correspond à la valeur fournie.

Les handlers

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_intro.html#handlers-running-operations-on-change

Les handlers permettent de lancer des opérations, comme relancer un service, lors de changements.

Un module, étant idempotent, un playbook peut détecter qu'il y a eu un changement significatif sur un système distant, et donc déclencer une opération en réaction à ce changement. Une notification est envoyée à la fin d'un bloc de tâche du playbook, et l'opération en réaction ne sera déclenchée qu'une seule fois même si plusieurs tâches différentes envoient cette notification.

Par exemple, plusieurs tâches peuvent indiquer que le service httpd nécessite une relance à cause d'un changement dans ses fichiers de configuration. Mais le service ne sera redémarré qu'une seule fois pour éviter les multiples démarrages non nécessaires.

```
- name: template configuration file
  template: src=modele-site.j2 dest=/etc/httpd/sites-availables/site-test.conf
  notify:
     - restart memcached
     - restart httpd
```

Un handler est une sorte de tâche référencé par un nom unique global :

- Il est activé par ou un plusieurs notifiers.
- Il ne se lance pas immédiatement, mais attend que toutes les tâches soient complètes pour s'exécuter.

Exemple de handlers

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
    - name: restart httpd
      service: name=httpd state=restarted
```

Depuis la version 2.2 d'Ansible, les handlers peuvent se mettre directement à l'écoute des tâches :

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
      listen: "restart web services"
    - name: restart apache
      service: name=apache state=restarted
      listen: "restart web services"

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

Les rôles

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_reuse_roles.html

Un rôle Ansible est une unité favorisant la réutilisabilité des playbooks.

Un squelette de rôle, servant comme point de départ du développement d'un rôle personnalisé, peut être généré par la commande `ansible-galaxy` :

```
$ ansible-galaxy init claranet
```

La commande aura pour effet de générer l'arborescence suivante pour contenir le rôle `claranet` :

```
$ tree claranet
claranet/
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Les rôles permettent de s'affranchir totalement de l'inclusion des fichiers. Plus besoin de spécifier les chemins d'accès aux fichiers, les directives `include` dans les playbook. Il suffit de spécifier une tâche, ansible s'occupe des inclusions.

La commande ansible-galaxy

La commande `ansible-galaxy` gère des rôles en utilisant le site galaxy.ansible.com.

Syntaxe de la commande ansible-galaxy

```
ansible-galaxy [import|init|install|login|remove|...]
```

Table 6. Sous-commandes de la commande ansible-galaxy

Sous-commandes	Observations
`install`	installe un rôle
`remove`	retire un ou plusieurs rôles
`init`	génère un squelette de nouveau rôle
`import`	importe un rôle depuis le site web galaxy. Nécessite un login au préalable.

Les tâches asynchrones

Plus d'informations sur http://docs.ansible.com/ansible/latest/playbooks_async.html

Par défaut, les connexions SSH vers les hosts restent ouvertes durant l'exécution des différentes tâches d'un playbook sur l'ensemble des noeuds.

Cela peut poser quelques problèmes, notamment :

- si la durée d'exécution de la tâche est supérieure au timeout de la connexion SSH
- si la connexion est interrompue durant l'action (reboot du serveur par exemple)

Dans ce cas, il faudra basculer en mode asynchrone et spécifier un temps maximum d'exécution ainsi que la fréquence (par défaut à 10s) avec laquelle vous allez vérifier le status de l'hôte.

En spécifiant une valeur de poll à 0, Ansible va exécuter la tâche et continuer sans se soucier du résultat.

Voici un exemple mettant en oeuvre les tâches asynchrones, qui permet de relancer un serveur et d'attendre que le port 22 soit de nouveau joignable :

```
# On attend 2s et on lance un reboot
- name: Reboot system
  shell: sleep 2 && shutdown -r now "Ansible reboot triggered"
  async: 1
  poll: 0
  ignore_errors: true
  become: true
  changed_when: False

  # On attend que ça revienne
  - name: Waiting for server to restart (10 mins max)
    wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
      delay: 30
      state: started
      timeout: 600
    delegate_to: localhost
```

Connexion à une instance Cloud Amazon ECS

Lors de la création d'une instance Amazon, une clef privée est créée et téléchargée sur le poste local.

Ajout de la clef dans l'agent SSH :

```
ssh-add path/to/fichier.pem
```

Lancement des `facts` sur les serveurs aws :

```
ansible aws --user=ec2-user --become -m setup
```

Pour une image ubuntu, il faudra utiliser l'utilisateur ubuntu :

```
ansible aws --user=ubuntu --become -m setup
```

Ansistrano

🎓 Objectifs

- ☐ Mettre en oeuvre Ansistrano ;
- ☐ Configurer Ansistrano ;
- ☐ Utiliser des dossiers et fichiers partagés entre versions déployées ;
- ☐ Déployer différentes versions d'un site depuis git ;
- ☐ Réagir entre les étapes de déploiement.

Introduction

Ansistrano est un rôle Ansible pour déployer facilement des applications PHP, Python, etc. Il se base sur le fonctionnement de Capistrano

Ansistrano nécessite pour fonctionner :

- Ansible sur la machine de déploiement,
- `rsync` ou `git` sur la machine cliente.

Il peut télécharger le code source depuis `rsync`, `git`, `scp`, `http`, `S3`, ...

Dans le cadre de notre exemple de déploiement, nous allons utiliser le protocole `git`.

Ansistrano déploie les applications en suivant ces 5 étapes :

- `Setup` : création de la structure de répertoire pour accueillir les releases
- `Update Code` : téléchargement de la nouvelle release sur les cibles
- `Symlink Shared` et `Symlink` : après avoir déployé la nouvelle release, le lien symbolique `current` est modifié pour pointer vers cette nouvelle release
- `Clean Up` : pour faire un peut de nettoyage (suppression des anciennes versions)

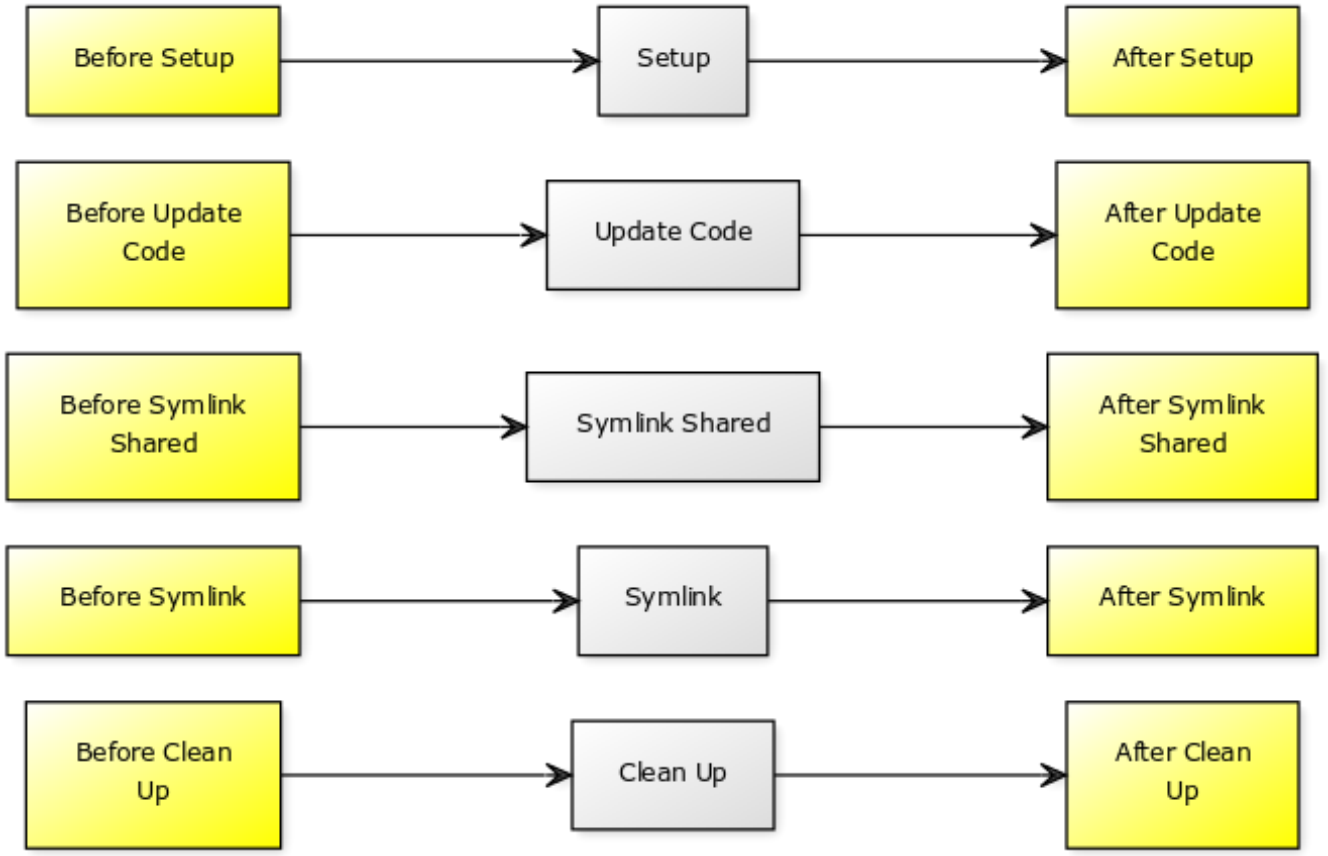


Figure 3. Etapes d'un déploiement

Le squelette d'un déploiement avec ansistrano ressemble à :

Squelette d'un déploiement avec ansistrano

```
/var/www/site/
├── current -> ./releases/20180821070043Z
├── releases
│   └── 20180821070043Z
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       └── REVISION
├── repo
└── shared
    ├── css/
    └── img/
```

Vous retrouverez toute la documentation ansistrano sur son dépôt Github.

Module 1 : Prise en main de la plateforme

Exercice 1.1 : Déploiement de la plateforme

Vous allez travailler sur 2 VM :

- La VM de gestion :
  - Vous devrez installer ansible et déployer le rôle `ansistrano-deploy`
- La VM cliente :
  - Cette VM n'a aucun logiciel spécifique.
  - Vous devrez installer Apache et déployer le site du client

Module 2 : Déployer le serveur Web

Exercice 2.1 : Utiliser le rôle `geerlingguy.apache` pour configurer le serveur

Pour gérer notre serveur web, nous allons utiliser le rôle `geerlingguy.apache` qu'il faut installer sur le serveur :

```
[ansiblesrv] $ ansible-galaxy install geerlingguy.apache
```

Une fois le rôle installé, nous allons pouvoir créer la première partie de notre playbook, qui va :

- Installer Apache,
- Créer un dossier cible pour notre `vhost`,
- Créer un `vhost` par défaut,
- Démarrer ou redémarrer Apache.

Considérations techniques :

- Nous allons déployer notre site dans le dossier `/var/www/site/`.
- Comme nous le verrons plus loin, `ansistrano` va créer un lien symbolique `current` vers le dossier de la release en cours.
- Le code source à déployer contient un dossier `html` sur lequel le vhost devra pointer. Sa `DirectoryIndex` est `index.htm`.
- Le déploiement se faisant par `git`, le paquet sera installé.

La cible de notre vhost sera donc : `/var/www/site/current/html`.

Playbook de configuration du serveur playbook-config-serveur.yml

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
        documentroot: "{{ dest }}current/html"

  tasks:

    - name: create directory for website
      file:
        path: /var/www/site/
        state: directory
        mode: 0755

    - name: install git
      package:
        name: git
        state: latest

  roles:
    - { role: geerlingguy.apache }
```

Le playbook peut être appliqué au serveur :

```
[ansiblesrv] $ ansible-playbook -i hosts playbook-config-serveur.yml
```

Notez l'exécution des tâches suivantes :

```
TASK [geerlingguy.apache : Ensure Apache is installed on RHEL.] ****************
TASK [geerlingguy.apache : Configure Apache.] **********************************
TASK [geerlingguy.apache : Add apache vhosts configuration.] *******************
TASK [geerlingguy.apache : Ensure Apache has selected state and enabled on boot.] ***
RUNNING HANDLER [geerlingguy.apache : restart apache] **************************
```

Le rôle `geerlingguy.apache` nous facililte grandement la tâche en s'occupant de l'installation et la configuration d'Apache.

Module 3 : Déployer le logiciel

Notre serveur étant maintenant configuré, nous allons pouvoir déployer l'applicatif.

Pour cela, nous allons utiliser le rôle `ansistrano.deploy` dans un second playbook dédié au déploiement applicatif (pour plus de lisibilité).

Exercice 3.1 : Installer le rôle `ansistrano-deploy`

```
[ansiblesrv] $ ansible-galaxy install ansistrano.deploy
```

Exercice 3.2 : Utiliser le rôle `ansistrano-deploy` pour déployer le logiciel

Les sources du logiciel se trouvent dans le dépôt :

- sur framagit https://framagit.org/alemorvan/demo-ansible.git accessible depuis internet.

Nous allons créer un playbook `playbook-deploy.yml` pour gérer notre déploiement :

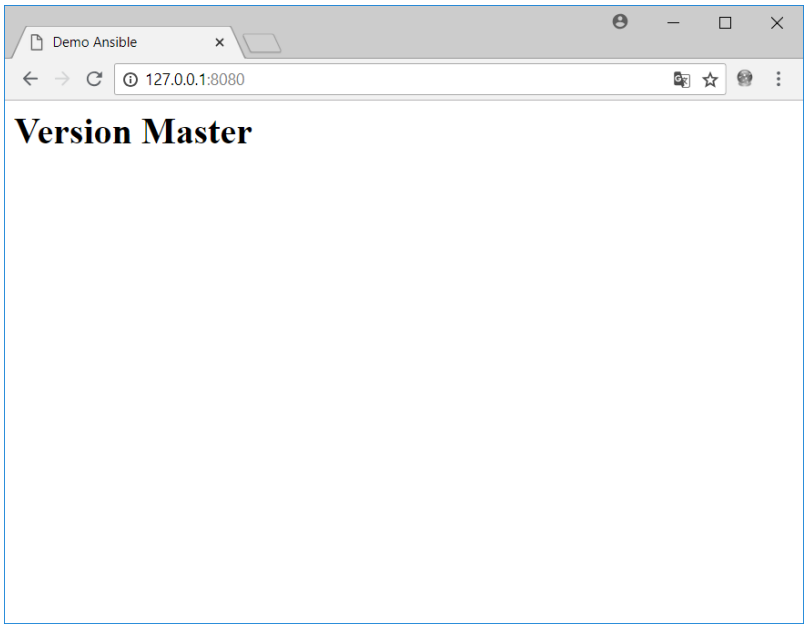
```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"

  roles:
     - { role: ansistrano.deploy }
```

```
[ansiblesrv] $ ansible-playbook -i hosts playbook-deploy.yml
```

Exercice 3.3 : Visualiser le résultat dans un navigateur

Une redirection du port `http` a été mis en place pour accéder depuis l'extérieur à votre serveur.

- Ouvrir un navigateur web vers l'url http://@IPPUBLIQUE:PORTREDIR/ :



Et voir apparaître le déploiement fonctionnel de la branche `master` de notre dépôt.

Exercice 3.4 : Vérification sur le serveur

- Se connecter en ssh sur le serveur client :

```
[ansiblesrv] $ ssh ansible@192.168.10.11
```

- Faire un `tree` sur le repertoire `/var/www/site/` :

```
[ansiblecli] $ tree /var/www/site/
/var/www/site/
├── current -> ./releases/20180821070043Z
├── releases
│   └── 20180821070043Z
│       ├── html
│       │   └── index.htm
│       └── REVISION
├── repo
│   └── html
│       └── index.htm
└── shared
```

Notez :

- le lien symbolique `current` vers la release `./releases/20180821070043Z`
- la présence d'un dossier `shared`
- la présence du dépôt git dans `./repo/`

- Depuis le serveur ansible, relancer **3** fois le déploiement, puis vérifier sur le client.

```
[ansiblecli] $ tree /var/www/site/
/var/www/site/
├── current -> ./releases/20180821112348Z
├── releases
│   ├── 20180821070043Z
│   │   ├── html
│   │   │   └── index.htm
│   │   └── REVISION
│   ├── 20180821112047Z
│   │   ├── html
│   │   │   └── index.htm
│   │   └── REVISION
│   └── 20180821112100Z
│       ├── html
│       │   └── index.htm
│       └── REVISION
│   ├── 20180821112348Z
│   │   ├── html
│   │   │   └── index.htm
│   │   └── REVISION
├── repo
│   └── html
│       └── index.htm
└── shared
```

Notez :

- `ansistrano` a conservé les 4 dernières releases,
- le lien `current` pointe maintenant vers la dernière release exécutée

Module 4 : Ansistrano

Exercice 4.1 : Limiter le nombre de releases

La variable `ansistrano_keep_releases` permet de spécifier le nombre de releases à conserver.

- En utilisant la variable `ansistrano_keep_releases`, ne conserver que 3 releases du projet. Vérifier.

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3

  roles:
     - { role: ansistrano.deploy }
```

```
---
[ansiblesrv] $ ansible-playbook -i hosts playbook-deploy.yml
```

Sur le client :

```
[ansiblecli] $ tree /var/www/site/
/var/www/site/
├── current -> ./releases/20180821113223Z
├── releases
│   └── 20180821112100Z
│       ├── html
│       │   └── index.htm
│       └── REVISION
│   ├── 20180821112348Z
│   │   ├── html
│   │   │   └── index.htm
│   │   └── REVISION
│   └── 20180821113223Z
│       ├── html
│       │   └── index.htm
│       └── REVISION
├── repo
│   └── html
│       └── index.htm
└── shared
```

Exercice 4.2 : Utiliser des `shared_paths` et `shared_files`

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"

  roles:
     - { role: ansistrano.deploy }
```

Sur le client, créer le fichier `logs` dans le répertoire `shared` :

```
sudo touch /var/www/site/shared/logs
```

Puis lancer l'exécution du job :

```
TASK [ansistrano.deploy : ANSISTRANO | Ensure shared paths targets are absent]
*****************************************************
ok: [192.168.10.11] => (item=img)
ok: [192.168.10.11] => (item=css)
ok: [192.168.10.11] => (item=logs/log)

TASK [ansistrano.deploy : ANSISTRANO | Create softlinks for shared paths and files]
*************************************************
changed: [192.168.10.11] => (item=img)
changed: [192.168.10.11] => (item=css)
changed: [192.168.10.11] => (item=logs)
```

Sur le client :

```
[ansiblecli] $  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20180821120131Z/
├── releases
│   ├── 20180821112348Z/
│   │   ├── html/
│   │   │   └── index.htm
│   │   └── REVISION
│   └── 20180821113223Z/
│       ├── html/
│       │   └── index.htm
│       └── REVISION
│   └── 20180821120131Z/
│       ├── css -> ../../shared/css/
│       ├── html
│       │   └── index.htm
│       ├── img -> ../../shared/img/
│       ├── logs -> ../../shared/logs
│       └── REVISION
├── repo/
│   └── html
│       └── index.htm
└── shared/
    ├── css/
    ├── img/
    └── logs
```

Notez que la dernière release contient :

- Un répertoire `css`, un répertoire `img`, et un fichier `logs`
- Des liens symboliques :
  - du dossier `/var/www/site/releases/css/` vers le dossier `../../shared/css/`.
  - du dossier `/var/www/site/releases/img/` vers le dossier `../../shared/img/`.
  - du fichier `/var/www/site/releases/logs/` vers le fichier `../../shared/logs`.

Dès lors, les fichiers contenus dans ces 2 dossiers et le fichier `logs` sont toujours accessibles via les chemins suivants :

- `/var/www/site/current/css/`,
- `/var/www/site/current/img/`,
- `/var/www/site/current/logs`,

mais surtout ils seront conservés d'une release à l'autre.

Exercice 4.3 : Utiliser un sous répertoire du dépôt pour le déploiement

Dans notre cas, le dépôt contient un dossier `html`, qui contient les fichiers du site.

- Pour éviter ce niveau supplémentaire de répertoire, utiliser la variable `ansistrano_git_repo_tree` en précisant le path du sous répertoire à utiliser. Ne pas oublier de modifier la configuration d'apache pour prendre en compte ce changement !

Playbook de configuration du serveur playbook-config-serveur.yml

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    apache_global_vhost_settings: |
      DirectoryIndex index.php index.htm
    apache_vhosts:
      - servername: "website"
        documentroot: "{{ dest }}current/" ①

  tasks:

    - name: create directory for website
      file:
        path: /var/www/site/
        state: directory
        mode: 0755

    - name: install git
      package:
        name: git
        state: latest

  roles:
    - { role: geerlingguy.apache }
```

① Modifier cette ligne

Playbook de déploiement playbook-deploy.yml

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html' ①

  roles:
     - { role: ansistrano.deploy }
```

① Modifier cette ligne

- Vérifier sur le client :

```
[ansiblecli] $  tree -F /var/www/site/
/var/www/site/
├── current -> ./releases/20180821120131Z/
├── releases
│   └── 20180821113223Z/
│   │   ├── html/
│   │   │   └── index.htm
│   │   └── REVISION
│   └── 20180821120131Z/
│   │   ├── css -> ../../shared/css/
│   │   ├── html
│   │   │   └── index.htm
│   │   ├── img -> ../../shared/img/
│   │   ├── logs -> ../../shared/logs
│   │   └── REVISION
│   └── 20180821130124Z/
│       ├── css -> ../../shared/css/
│       ├── img -> ../../shared/img/
│       ├── index.htm  ①
│       ├── logs -> ../../shared/logs
│       └── REVISION
├── repo/
│   └── html
│       └── index.htm
└── shared/
    ├── css/
    ├── img/
    └── logs
```

① Notez l'absence du dossier `html`

Module 5 : La gestion des branches ou des tags git

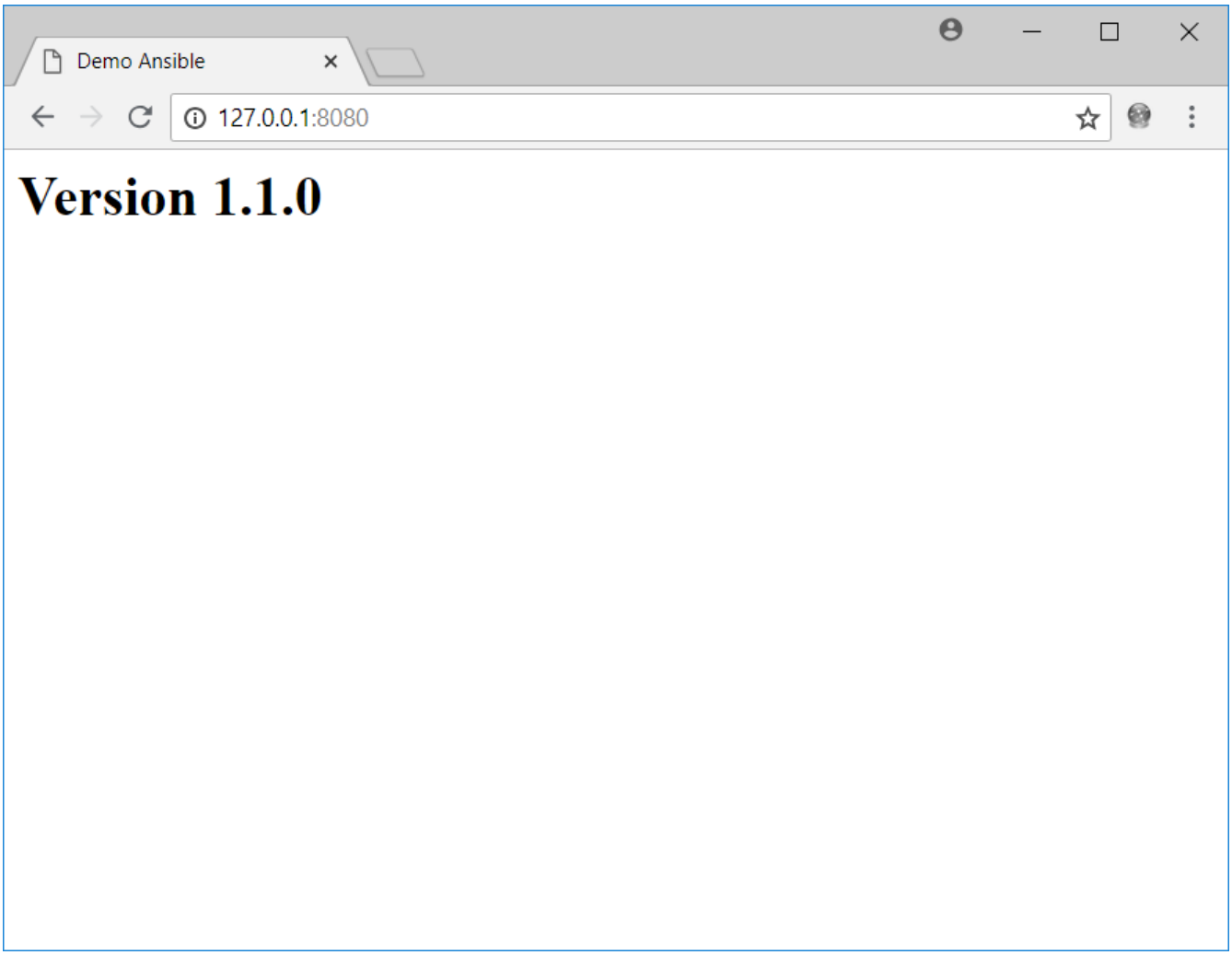
La variable `ansistrano_git_branch` permet de préciser une `branch` ou un `tag` à déployer.

Exercice 5.1 : Déployer une branche

- Déployer la branche `releases/v1.1.0` :

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'releases/v1.1.0'

  roles:
     - { role: ansistrano.deploy }
```

Vous pouvez vous amuser, durant le déploiement, à rafraichir votre navigateur, pour voir en 'live' le changement s'effectuer.

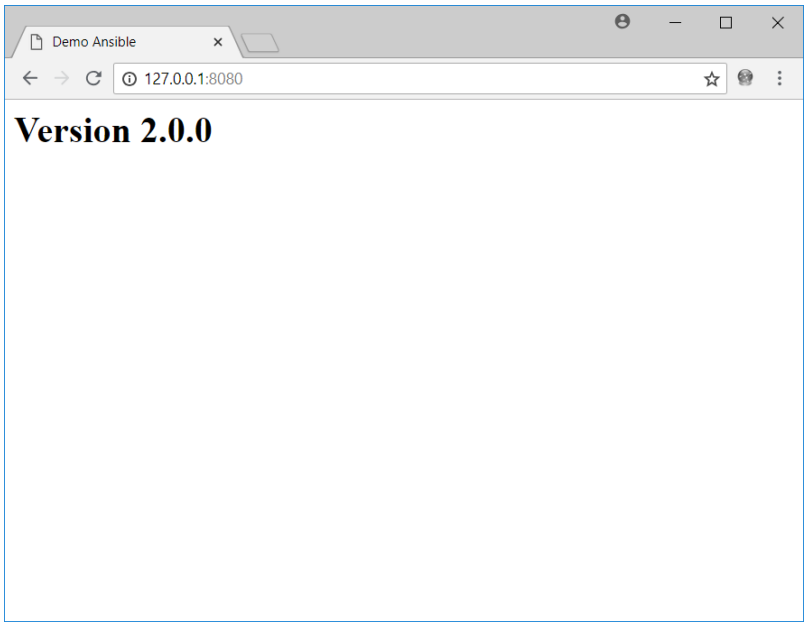


Exercice 5.2 : Déployer un tag

- Déployer le tag `v2.0.0` :

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "log"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'

  roles:
     - { role: ansistrano.deploy }
```

Vous pouvez vous amuser, durant le déploiement, à rafraichir votre navigateur, pour voir en 'live' le changement s'effectuer.



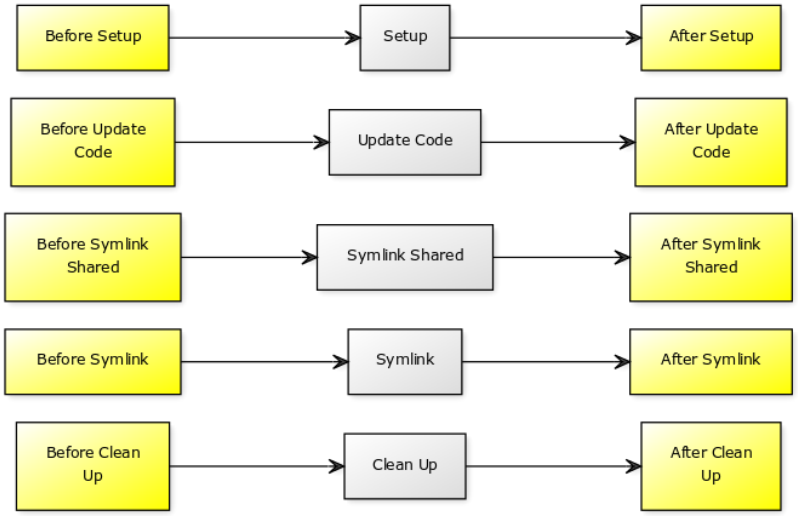
Module 6 : Actions entre les étapes de déploiement

Un déploiement avec Ansistrano respecte les étapes suivantes :

- `Setup`
- `Update Code`
- `Symlink Shared`

- Symlink
- Clean Up

Il est possible d'intervenir avant et après chacune de ses étapes.



Un playbook peut être inclu par les variables prévues à cet effet :

- ansistrano_before_<task>_tasks_file
- ou ansistrano_after_<task>_tasks_file

Exercice 6.1 : Envoyer un mail en début de MEP

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-setup-
tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Le fichier deploy/before-setup-tasks.yml

```
---
- name: Envoyer un mail
  mail:
    subject: Debut de MEP sur {{ ansible_hostname }}.
  delegate_to: localhost
```

```
TASK [ansistrano.deploy : include]
*********************************************************************************
included: /home/ansible/deploy/before-setup-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : Envoyer un mail]
*********************************************************************************
ok: [192.168.10.11 -> localhost]
```

```
[root] # mailx
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N  1 root@localhost.local  Tue Aug 21 14:41  28/946   "Debut de MEP sur localhost."
```

Exercice 6.2 : Redémarrer apache en fin de MEP

```
---
- hosts: ansiblecli
  become: yes
  vars:
    dest: "/var/www/site/"
    ansistrano_deploy_via: "git"
    ansistrano_git_repo: https://framagit.org/alemorvan/demo-ansible.git
    ansistrano_deploy_to: "{{ dest }}"
    ansistrano_keep_releases: 3
    ansistrano_shared_paths:
      - "img"
      - "css"
    ansistrano_shared_files:
      - "logs"
    ansistrano_git_repo_tree: 'html'
    ansistrano_git_branch: 'v2.0.0'
    ansistrano_before_setup_tasks_file: "{{ playbook_dir }}/deploy/before-setup-
tasks.yml"
    ansistrano_after_symlink_tasks_file: "{{ playbook_dir }}/deploy/after-symlink-
tasks.yml"

  roles:
     - { role: ansistrano.deploy }
```

Le fichier deploy/after-symlink-tasks.yml

```
---
- name: restart apache
  service:
    name: httpd
    state: restarted
```

```
TASK [ansistrano.deploy : include]
*********************************************************************************
included: /home/ansible/deploy/after-symlink-tasks.yml for 192.168.10.11

TASK [ansistrano.deploy : restart apache]
*********************************************************************************
changed: [192.168.10.11]
```

Ordonnanceur centralisé Rundeck

🎓 Objectifs

☑ installer un serveur d'ordonnancement Rundeck ;
☑ créer un premier projet et une tâche simple.

Rundeck est un ordonnanceur centralisé open source (licence Apache) écrit en Java.

Depuis l'interface de Rundeck, il est possible de gérer les différents travaux (commandes ou scripts) à exécuter sur les serveurs distants.

Fonctionnalités :

- Ordonnancement des tâches selon un scénario ;
- Exécution de scripts/tâches distantes à la demande ou de manière planifiée ;
- Notifications ;
- Interface CLI (ligne de commande) et API.

Rundeck peut être intégré aux autres outils de gestion de configuration comme Puppet, Ansible et d'intégration continue comme Jenkins, etc.

La connexion avec les hôtes clients est gérée en SSH.

Installation

Rundeck utilisant le protocle SSH pour se connecter aux systèmes distants, un compte avec les droits sudo est nécessaire sur chacun des stations clientes.

- sur Debian 8 (Jessie) :

Rundeck étant écrit en Java, l'installation du JDK est nécessaire :

```
# apt-get install openjdk-7-jdk
```

Rundeck peut être téléchargé puis installé :

```
# wget http://dl.bintray.com/rundeck/rundeck-deb/rundeck-2.6.7-1-GA.deb
# dpkg -i ./rundeck-2.6.7-1-GA.deb
```

- Sur CentOS 7 :

Rundeck étant écrit en Java, l'installation du JDK est nécessaire :

```
# yum install java-1.8.0
```

Rundeck peut être téléchargé puis installé :

```
# rpm -Uvh http://repo.rundeck.org/latest.rpm
# yum install rundeck
```

Configuration

Par défaut, Rundeck n'est configuré en écoute que sur l'adresse de boucle interne (`localhost`).

Si vous ne souhaitez pas mettre en place un proxy inverse (Apache, Nginx ou HAProxy), éditez les fichiers `/etc/rundeck/framework.properties` et `/etc/rundeck/rundeck-config.properties` et remplacer `localhost` par l'adresse IP du serveur, pour mettre en écoute Rundeck sur son interface réseau :

Modification du fichier /etc/rundeck/framework.properties

```
framework.server.url = http://xxx.xxx.xxx.xxx:4440
```

Modification du fichier /etc/rundeck/rundeck-config.properties

```
grails.serverURL=http://xxx.xxx.xxx.xxx:4440
```

Remplacer `xxx.xxx.xxx.xxx` par l'adresse IP publique du serveur.

Rundeck est ensuite lancé par systemctl :

```
# systemctl start rundeckd
# systemctl enable rundeckd
```

L'interface d'administration de Rundeck est accessible depuis un navigateur internet à l'adresse `http://your_server:4440`.

Utiliser RunDeck

Le compte par défaut pour se connecter à l'interface web est `admin` (mot de passe : `admin`). Pensez à changer le mot de passe le plus rapidement possible.

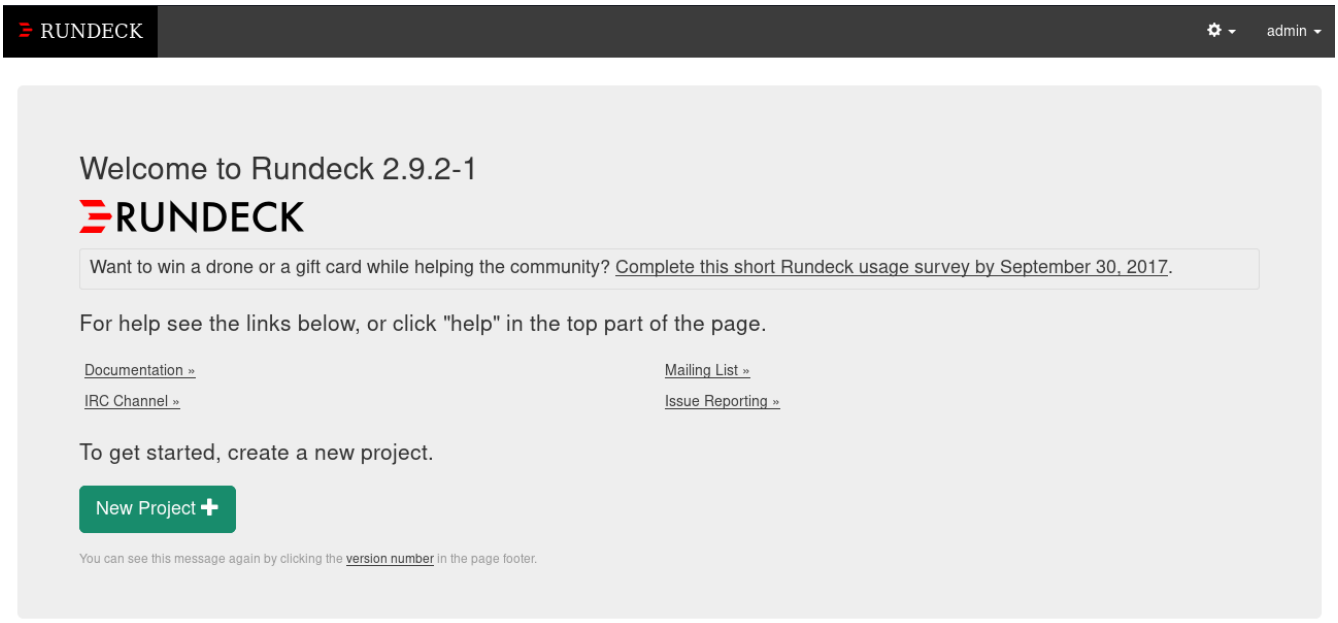


Figure 4. La page d'accueil de RunDeck

Créer un projet

Un nouveau projet peut être ajouté en cliquant sur le lien `Nouveau projet`.

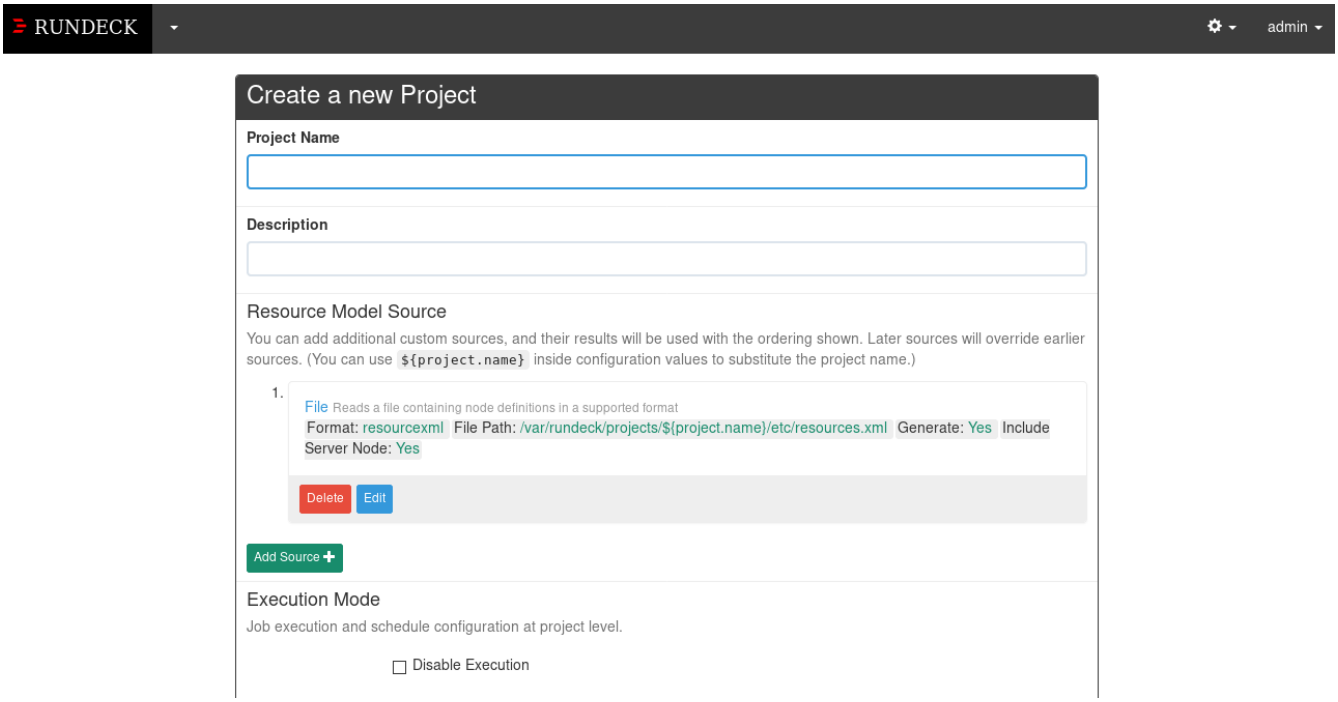


Figure 5. L'assistant de création de nouveau projet Rundeck

Vous devez fournir au minimum un nom de projet.

Dans la section `Resource Model Source`, cliquer sur le bouton `Edit` et choisir `Require File Exists `, puis cliquer sur `Save`.

Dans la section `Default Node Executor`, choisir `password` pour l'authentification par SSH (pour une meilleure gestion de la sécurité, l'utilisation d'une clef SSH est recommandée).

Cliquer sur `Create` pour créer le projet.

Créer une tâche

Le serveur est prêt à recevoir une première tâche. Cette tâche consiste en une connection SSH pour lancer une commande distante.

- Cliquer sur `Create a new job` et saisir un nom pour la tâche.

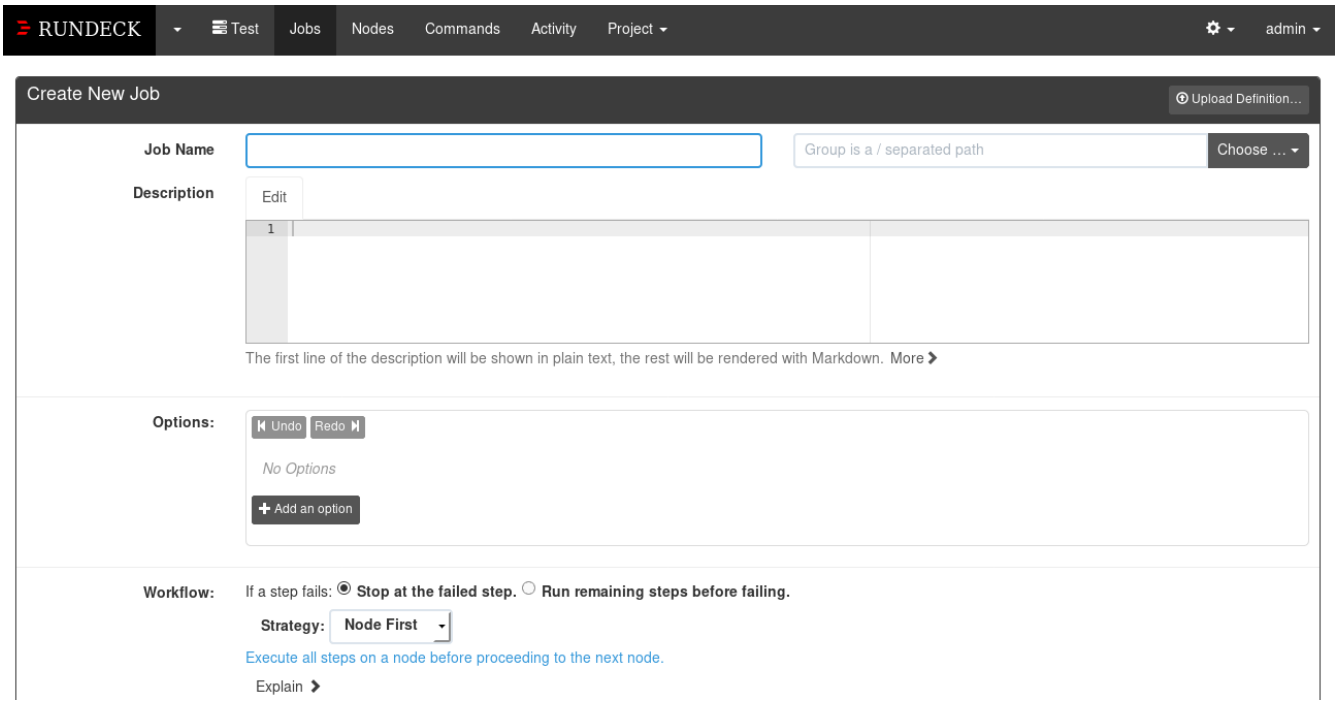
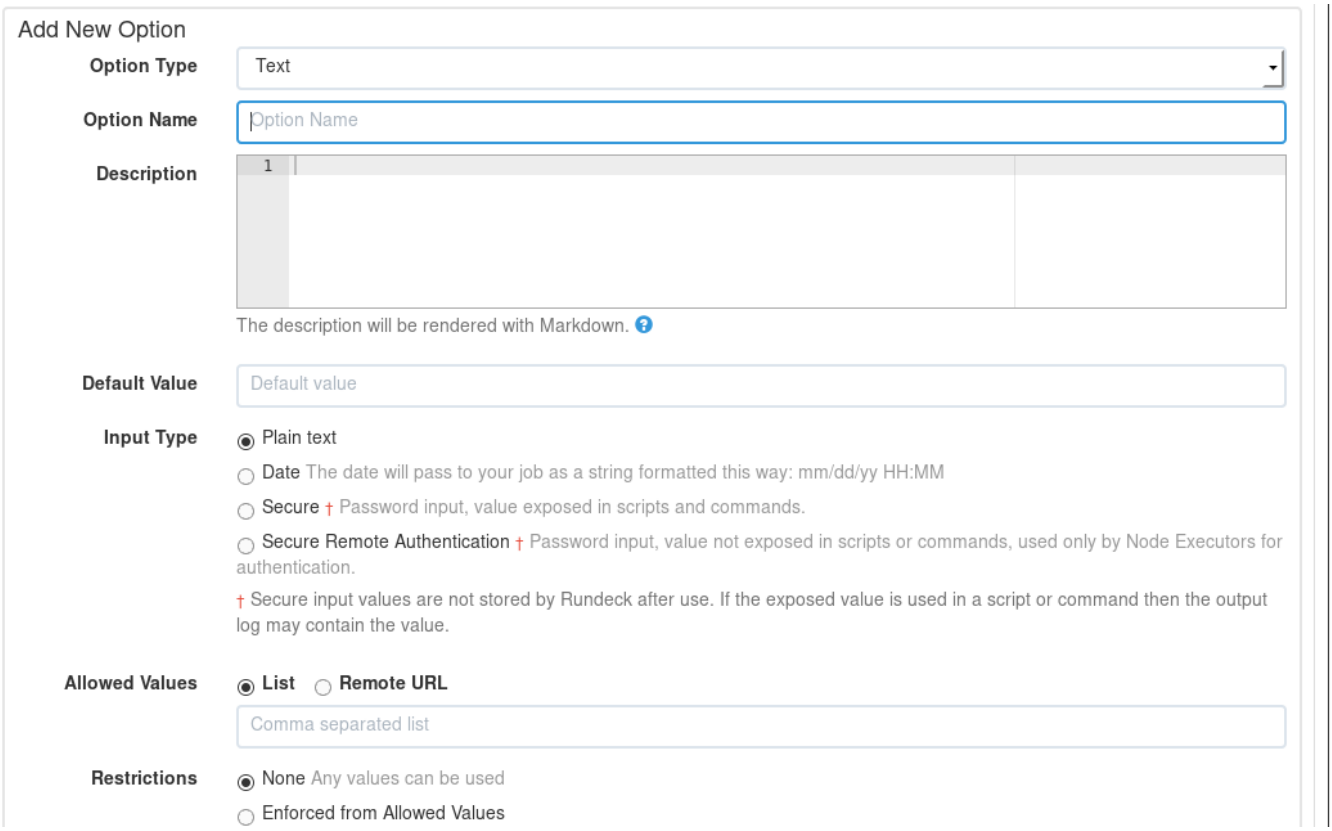


Figure 6. L'assistant de création de tâche de RunDeck

Il va falloir fournir à RunDeck le mot de passe de l'utilisateur distant permettant de se connecter au serveur ainsi que le mot de passe sudo pour lancer la commande. Pour cela :

- Ajouter une option

Cliquer sur `Add New Option`.



Donner un nom pour l'option (par exemple sshPassword) et une valeur par défaut qui correspond à votre mot de passe.

Dans le champ `Input Type`, choisir `Secure Remote Authentication` et mettre `Required` à `Yes`.

Répéter l'opération avec l'option `sudoPassword`.

Dans la section `Add a Step`, choisir `Command`. Fournir la commande dans le champ `Command`. Par exemple :

```
sudo "top -b -n 1 | head -n 5"
```

Cliquer sur `Save` puis `Create` pour finaliser la nouvelle tâche.

Pour appliquer cette tâche à un système distant (appelé noeud), éditer le fichier de configuration du noeud :

```
vi /var/rundeck/projects/your_project_name/etc/resources.xml
```

Ajouter à la ligne commençant par `localhost` : `ssh-authentication="password" ssh-password-option="option.sshPassword" sudo-command-enabled="true" sudo-password-option="option.sudoPassword"`, pour activer l'authentification par SSH et fournir les valeurs des options pour l'authentification.

Retourner dans l'interface web est executer la tâche.

Le serveur RunDeck est prêt à recevoir vos projets d'ordonnancement.

Serveur d'Intégration Continue Jenkins

🎓 Objectifs

☑ installer un serveur d'intégration continue Jenkins ;
☑ configurer l'accès au service par un mandataire proxy inverse ;
☑ sécuriser les accès au service ;
☑ créer une première tâche simple.

Jenkins est un outil Open Source d'**intégration continue** écrit en **Java**.

Interfacé avec des systèmes de gestion de version tel que Git, Subversion etc., il peut être utilisé pour compiler, tester et déployer des scripts shell ou des projets Ant, Maven, etc, selon des planifications ou des requêtes à des URLs.

Le principe de l'intégration continue est de vérifier, idéalement à chaque modification du code, la non-régression sur l'application des modifications apportées. Cette vérification systématique est primordiale pour une équipe de développeur : le projet est stable tout au long du développement et peut être livré à tout moment.

Le code source de l'application doit être :

- **partagé** avec un système de gestion versions ;
- **testé** automatiquement ;
- les modifications doivent être **poussées** régulièrement.

Ainsi, les problèmes d'intégrations peuvent être corrigés au plus tôt et la dernière version stable de l'application est connue.

Installation

Jenkins peut fonctionner :

- en mode **standalone** : avec le serveur web intégré ;
- en tant que **servlet** : avec le serveur applicatif tomcat.

En mode standalone

Installation de Jenkins :

- Sous debian :

Installation de la clé et ajout du dépot dans /etc/apt/sources.list :

```
# wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | # sudo apt-key add -
```

```
# deb http://pkg.jenkins-ci.org/debian binary/
```

Installation :

```
# apt-get update
# apt-get install jenkins
```

- Sous RedHat :

Installation du dépôt et ajout de la clé GPG :

```
# wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo
# sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key
```

Installation du paquet :

```
# yum update
# yum install java-1.8.0-openjdk jenkins
```

Jenkins peut maintenant être démarré :

```
# systemctl start jenkins
# systemctl enable jenkins
```

Jenkins est directement accessible à l'adressse :

- http://IPSERVEUR:8080/jenkins

En tant que servlet tomcat

Installation de tomcat :

- Sous debian :

```
apt-get install tomcat
```

- Sous redhat :

```
yum install java-1.8.0-openjdk tomcat
```

Téléchargement de l'application :

```
cd /var/lib/tomcat6/webapps
wget http://mirrors.jenkins-ci.org/war/latest/jenkins.war
```

Si tomcat est en cours de fonctionnement, l'application sera automatiquement déployée, sinon lancer/relancer tomcat.

Jenkins est accessible à l'adressse :

- http://IPSERVEUR:8080/jenkins

Installer Nginx

Un proxy inverse Nginx (mais Apache ou HAproxy sont de bonnes alternatives) va permettre d'accéder à l'application sur le port standard 80.

```
# yum -y install epel-release
# yum -y install nginx
# systemctl enable nginx
```

Créer un nouveau serveur dans la configuration de Nginx :

```
# vim /etc/nginx/conf.d/jenkins.conf
upstream jenkins{
    server 127.0.0.1:8080;
}

server{
    listen      80;
    server_name jenkins.formatux.fr;

    access_log  /var/log/nginx/jenkins.access.log;
    error_log   /var/log/nginx/jenkins.error.log;

    proxy_buffers 16 64k;
    proxy_buffer_size 128k;

    location / {
        proxy_pass  http://jenkins;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503
http_504;
        proxy_redirect off;

        proxy_set_header    Host            $host;
        proxy_set_header    X-Real-IP       $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto https;
    }

}
```

Lancer le serveur Nginx :

```
# systemctl start nginx
# systemctl enable nginx
```

Vous pouvez alternativement installer un proxy inversé Apache. Dans ce cas, le VHOST suivant pourra être utilisé :

```
<Virtualhost *:80>
    ServerName        jenkins.formatux.fr
    ProxyRequests     Off
    ProxyPreserveHost On
    AllowEncodedSlashes NoDecode

    <Proxy http://localhost:8080/*>
      Order deny,allow
      Allow from all
    </Proxy>

    ProxyPass         /  http://localhost:8080/ nocanon
    ProxyPassReverse  /  http://localhost:8080/
    ProxyPassReverse  /  http://jenkins.formatux.fr/
</Virtualhost>
```

Ouvrir les ports du parefeu :

- Sur un serveur en standalone sans proxy-inverse ou sous tomcat :

```
# firewall-cmd --zone=public --add-port=8080/tcp --permanent
```

Sur un serveur avec proxy inverse :

```
# firewall-cmd --zone=public --add-service=http --permanent
# firewall-cmd --reload
```

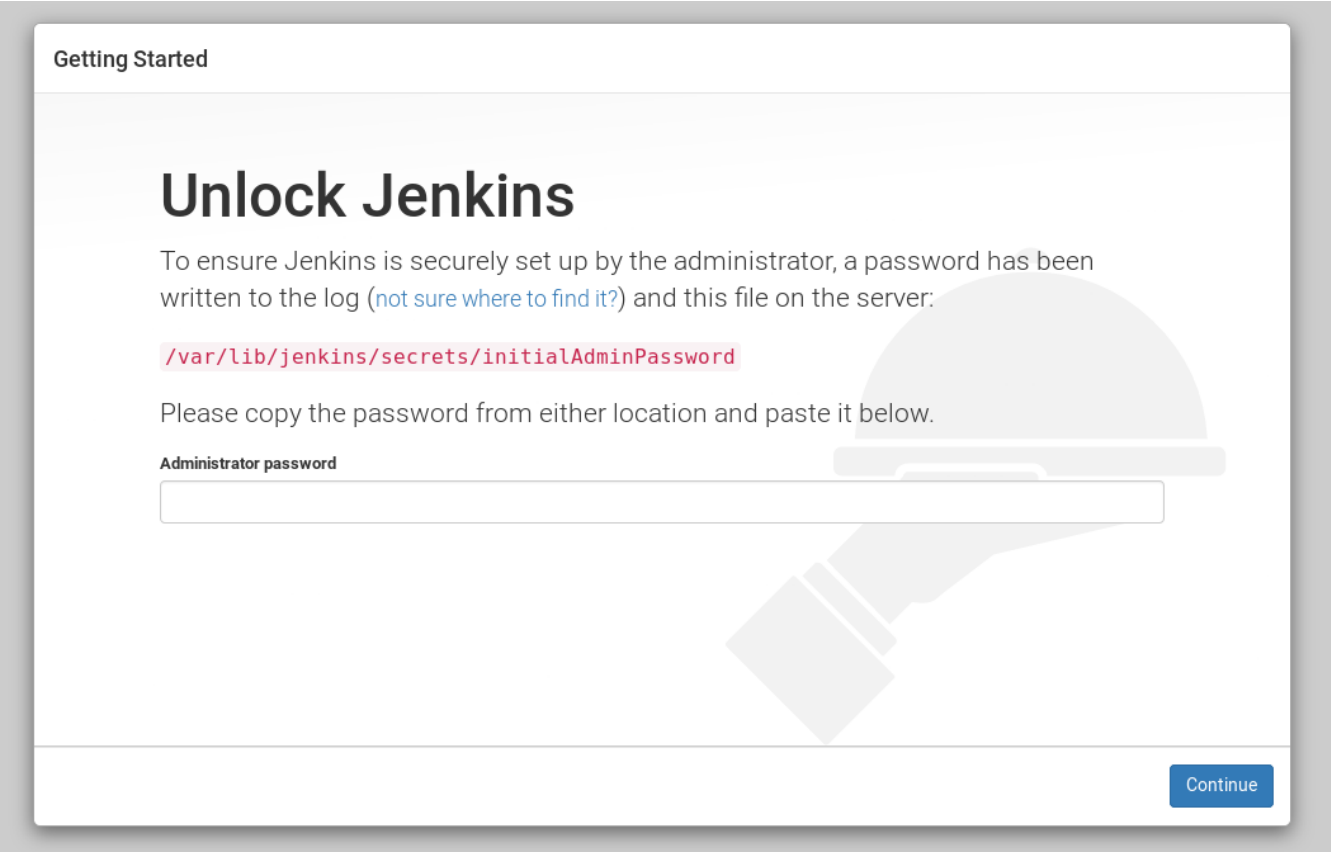
Autoriser Nginx à se connecter au serveur Jenkins d'un point de vue SELinux :

```
# setsebool httpd_can_network_connect 1 -P
```

Configuration de Jenkins

L'interface de gestion web du serveur d'intégration continue Jenkins est accessible à l'adresse `http://jenkins.formatux.fr` si le proxy inverse a été configuré ou à l'adresse `http://jenkins.formatux.fr:8080` dans les autres cas.

La page par défaut suivante s'affiche :



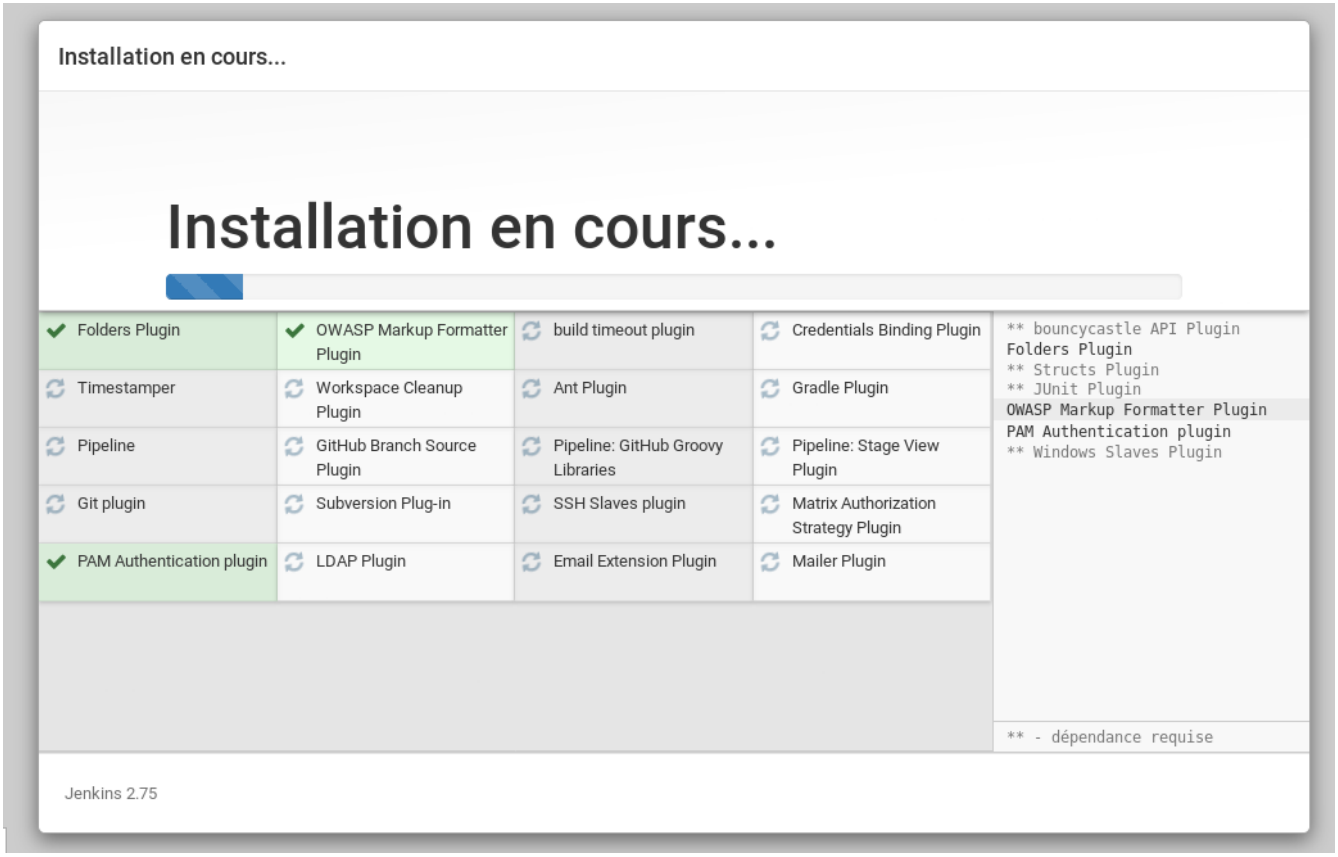
Cette page demande un mot de passe `admin` initial, qui a été généré lors de l'installation et qui est stocké dans le fichier `/var/lib/jenkins/secrets/initialAdminPassword`.

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

Utiliser le mot de passe pour ce connecter à l'interface de gestion.

L'assistant va ensuite demander quels plugins doivent être installés et proposer l'installation des plugins suggérés.

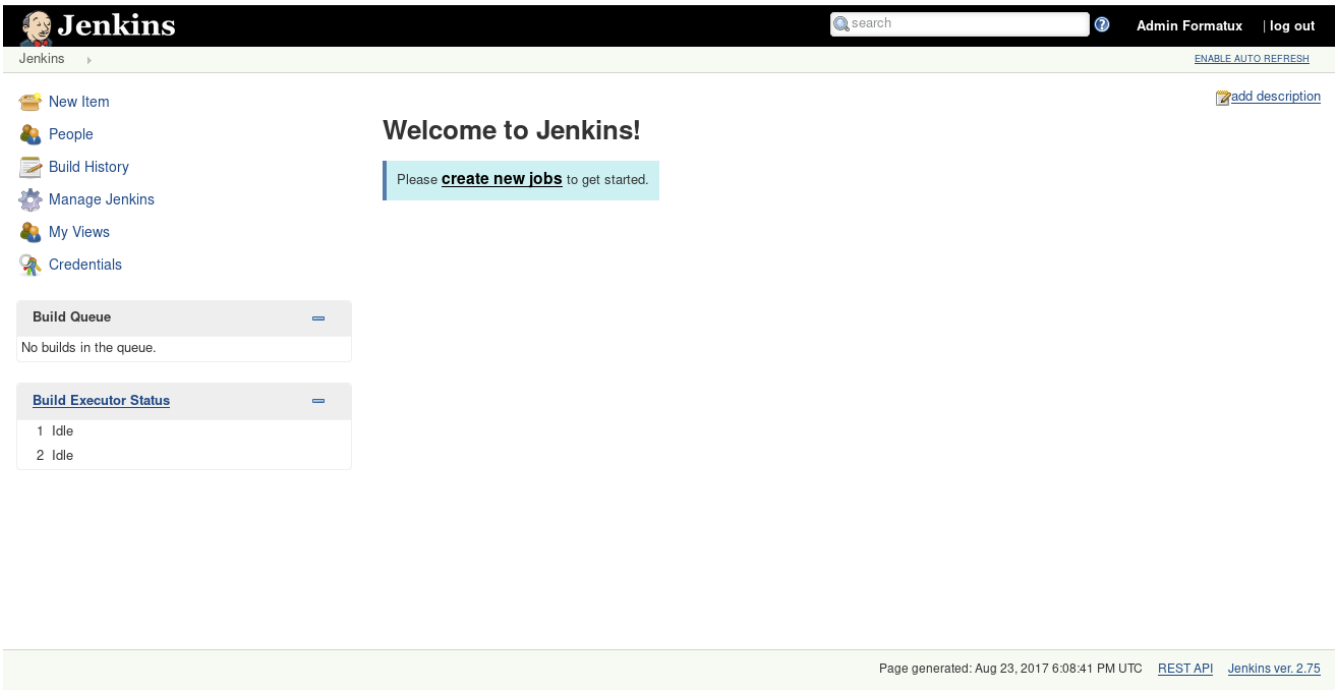
En choisissant `Installer les plugins suggérés`, tous les plugins nécessaire pour bien commencer seront installés :



L'étape suivante consiste à créer un utilisateur avec les droits d'administration pour l'accès à la

console de gestion :

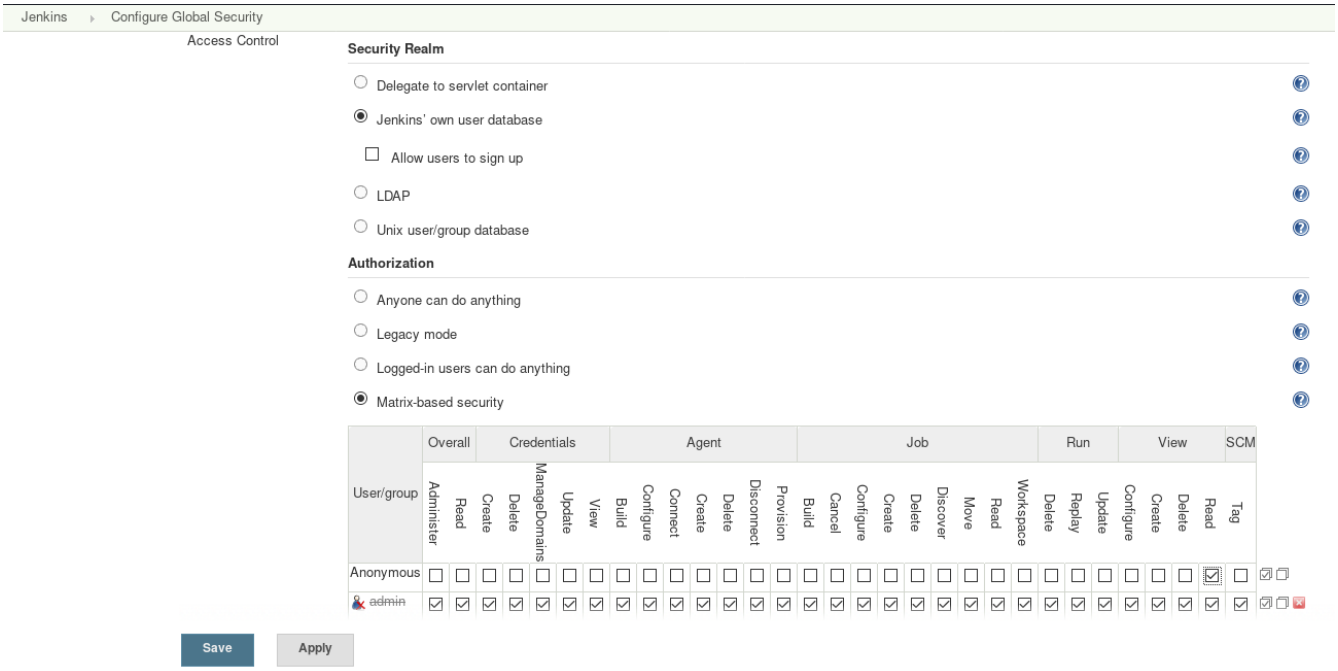
La configuration terminée, la console de gestion s'affiche :



La sécurité et la gestion des utilisateurs

Depuis l'interface de gestion, configurer les options de sécurité de Jenkins : cliquer sur Manage

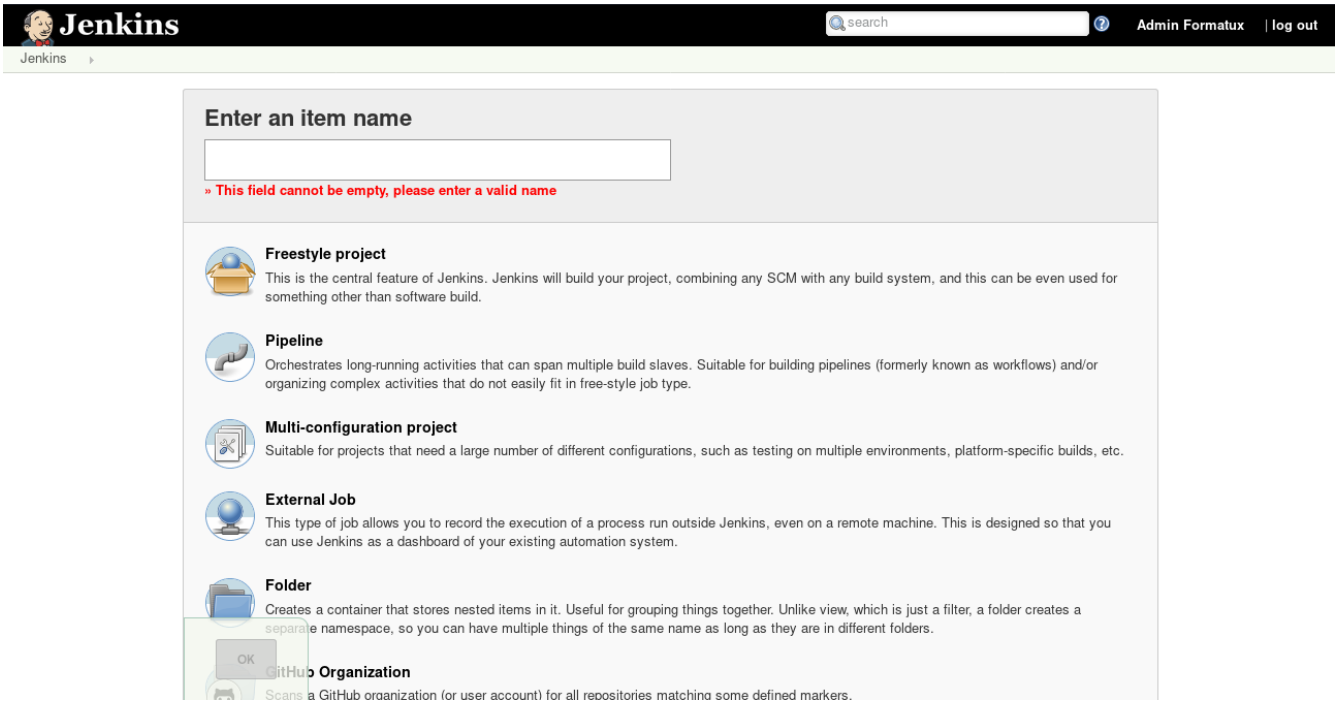
`Jenkins`, puis `Configure Global Security`.



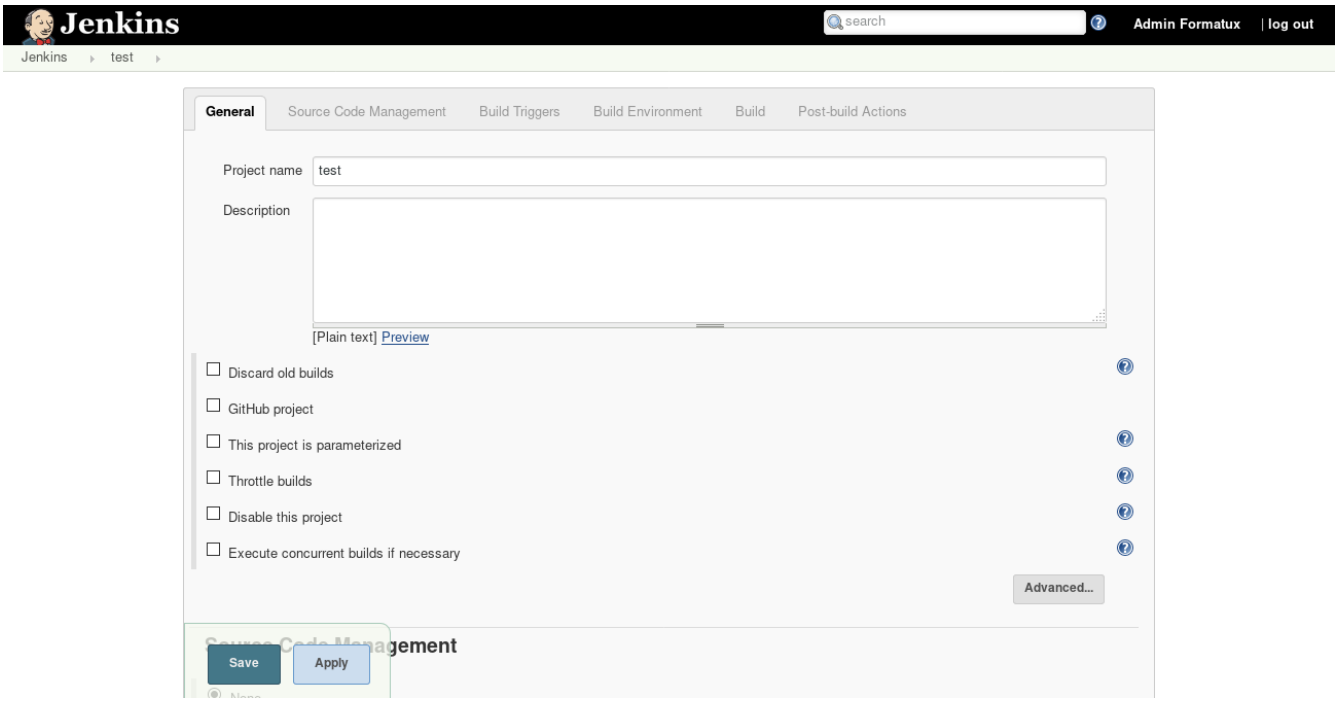
Plusieurs méthodes d'autorisations peuvent être utilisées. En sélectionnant `Matrix-based Security`, les droits des utilisateurs peuvent être gérés plus finement. Activer l'utilisateur `admin` et cliquer sur `Add`. Lui donner tous les droits en sélectionnant toutes les options. Donner à l'utilisateur `anonymous` uniquement les droits de lecture (`read`). Ne pas oublier de cliquer sur `Save`.

Ajouter une tâche d'automatisation simple

Dans l'interface de gestion, cliquer sur `Create new jobs`.



Saisir un nom pour la tâche, par exemple `test`. Choisir `Freestyle Project` et cliquer sur `OK`.

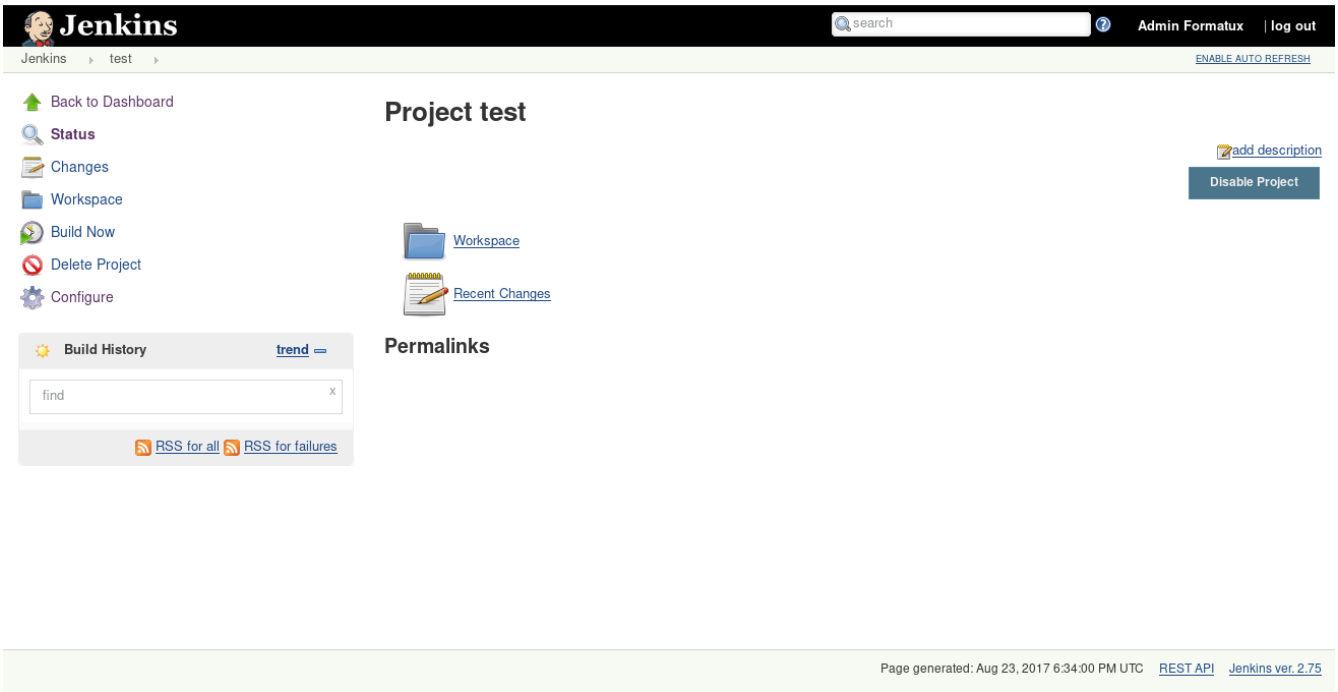


Aller dans l'onglet `Build`. Dans `Add build step`, selectionner l'option `Execute shell`.

Saisir la commande suivante dans le champ texte :

```
top -b -n 1 | head -n 5
```

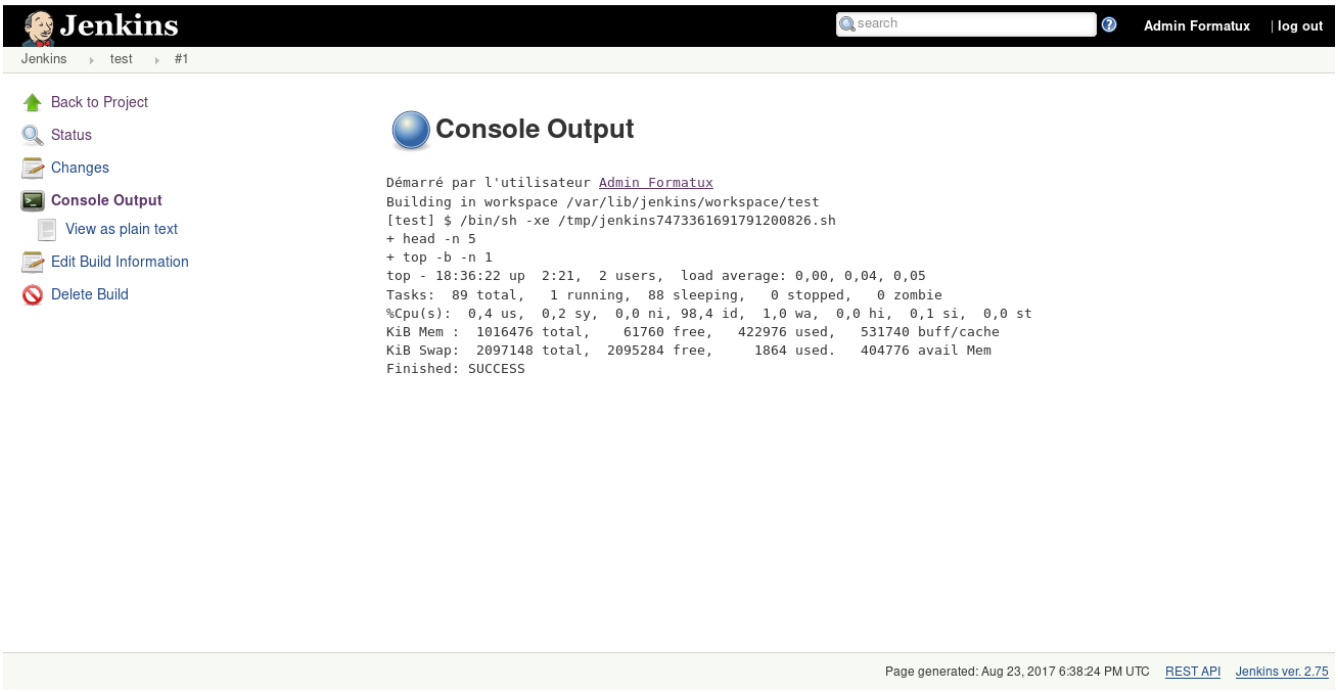
Cliquer sur `Save`.



Dans la page dédiée à la tâche `test`, cliquer sur `Build Now` pour executer la tâche `test`.

Une fois que la tâche a été executée, vous verrez l'historique du Build. Cliquer sur la première tâche

pour visualiser les résultats.



Sources

Source principale : howtoforge.

Asciidoc : Docs as Code

Introduction

La **Document as Code** (*Docs as Code*) est une philosophie de rédaction documentaire.

Pour favoriser la rédaction de documentations de qualité, les mêmes outils que ceux utilisés pour coder sont utilisés :

- gestionnaire de bugs,
- gestionnaire de version (`git`),
- revue de code,
- tests automatiques.

Plusieurs langages de balisage légers répondant à ces besoins sont apparus :

- le Markdown,
- le reStructuredText,
- l'Asciidoc.

> Le contenu avant la forme

Les avantages d'un tel dispositif sont nombreux :

- Le rédacteur se concentre uniquement sur le contenu et non sur la mise en forme comme c'est malheureusement trop souvent le cas avec les éditeurs de texte WYSIWYG (openoffice, word, etc.),
- Le même code source permet de générer des formats différents : pdf, html, word, confluence, epub, manpage, etc.
- Le travail collaboratif est grandement simplifié : code review, merge request, etc.
- Gestion des versions par branches ou tags git (pas une copie de fichier .doc).
- Edition de la documentation accessible à tous avec un simple éditeur de texte puisque la documentation est composée de fichiers plats (texte).

Le format asciidoc

Ce langage de balisage léger a été créé en 2002 !

Le processeur initial était développé en python mais à depuis été réécrit en ruby (https://asciidoctor.org/).

La syntaxe asciidoc a eu du mal à percer au bénéfice du markdown, mais sa grande lisibilité et le

lead de Dan Allen (https://twitter.com/mojavelinux) lui permettent de rattraper aujourd'hui son retard.

Voici un petit exemple de source asciidoc :

```
= Hello, AsciiDoc!
Doc Writer <doc@example.com>

An introduction to http://asciidoc.org[AsciiDoc].

== First Section

* item 1
* item 2
```

Comme vous pouvez le constater, le texte (malgré le balisage) reste lisible, la syntaxe est facilement assimilable et n'est pas trop lourde comparée à ce qui se faisait en latex.

> Use AsciiDoc for document markup. Really. It's actually readable by humans, easier to parse and way more flexible than XML.
>
> — Linus Torvalds

Une page référence les possibilités de la syntaxe asciidoc :

- https://asciidoctor.org/docs/asciidoc-syntax-quick-reference/

Compiler sa documentation

La documentation d'installation est disponible ici : https://asciidoctor.org/docs/install-toolchain/.

Comme pour un programme, la documentation nécessite une phase de compilation.

Après avoir installé l'environnement asciidoc, la commande suivante permet de compiler son document `.adoc` en `html5` (format par défaut) :

```
asciidoctor index.adoc
```

Une image docker existant, il est facile d'utiliser la CI d'un environnement tel que GitLab pour générer automatiquement la documentation à chaque commit, tag, etc.

```
build:
  stage: build
  image: asciidoctor/docker-asciidoctor
  script:
    - asciidoctor-pdf -a icons="font" -a lang="fr" -D public index.adoc
    only:
     - master
    artifacts:
      name: "compilation-pdf"
      paths:
        - public/
```

Une proposition d'environnement de travail

Atom (https://atom.io/) est un éditeur de code.

Après avoir installé Atom en suivant cette documentation : https://flight-manual.atom.io/getting-started/sections/installing-atom/, activer les packages suivants :

- `asciidoc-image-helper`
- `asciidoc-assistant`
- `asciidoc-preview`
- `autocomplete-asciidoc`
- `language-asciidoc`

Références

- http://www.writethedocs.org/guide/docs-as-code/
- https://www.technologies-ebusiness.com/enjeux-et-tendances/moi-code-madoc

Terraform : Infrastructure as Code

Introduction

Terraform est un produit de la société **HashiCorp** (*Terraform*, *Vault*, *Consul*, *Packer*, *Vagrant*), permettant de :

- **décrire** dans un langage humainement compréhensible l'infrastructure cible : la **HCL** (**H**ashiCorp **C**onfiguration **L**anguage),
- **versionner** les changements,
- **planifier** le déploiement,
- **créer** l'infrastructure.

Les changements apportés par une modification de code à l'infrastructure sont **prédictifs**.

> If it can be codified, it can be automated.

Le même outil permet de créer des ressources chez les plus grands fournisseurs d'infrastructure :

- AWS,
- GCP,
- Azure,
- OpenStack,
- VMware,
- etc.

L'infrastructure devient reproductible (intégration ⇒ preproduction ⇒ production) et facilement scalable. Les erreurs de manipulations humaines sont réduites.

Un exemple de création de ressouces :

```
# Une machine virtuelle
resource "digitalocean_droplet" "web" {
  name   = "tf-web"
  size   = "512mb"
  image  = "centos-7-5-x86"
  region = "sfo1"
}

/* Un enregistrement DNS
   en IPV4 type "A"
  */
resource "dnsimple_record" "hello" {
  domain = "example.com"
  name   = "test"
  value  = "${digitalocean_droplet.web.ipv4_address}"
  type   = "A"
}
```

Retrouvez plus d'informations sur le site https://www.terraform.io/, dans la documentation https://www.terraform.io/intro/index.html et dans l'espace de formation https://learn.hashicorp.com/terraform/getting-started/install.html.

La HCL

Le langage **H**ashiCorp **C**onfiguration **L**anguage est spécifique. Voici quelques points d'attention :

- Les commentaires sur une seule ligne commencent avec un `#`
- Les commentaires sur plusieurs lignes sont encadrés par `/*` et `*/`
- Les variables sont assignées avec la syntaxe `key = value` (aucune importance concernant les espaces). Les valeurs peuvent être des primitives `string`, `number` ou `boolean` ou encore une `list` ou une `map`.
- Les chaînes de caractères sont encadrées par des doubles-quotes.
- Les valeurs booléennes peuvent être `true` ou `false`.

Les providers

Terraform est utilisé pour gérer des ressources qui peuvent être des serveurs physiques, des VM, des équipements réseaux, des containeurs, mais aussi pourquoi pas des utilisateurs grafana, etc.

La liste des providers disponibles est consultable ici : https://www.terraform.io/docs/providers/index.html.

En voici quelques-uns :

- AWS,
- Azure,
- VMware vSphere
- OpenStack, CloudStack, OVH, DigitalOcean, etc.
- Gitlab,
- Datadog, PagerDuty,
- MySQL,
- Active Directory
- ...

Chaque provider venant avec ses propres ressources, il faut lire la doc !

Les actions

- `terraform init`

La première commande à lancer pour une nouvelle configuration qui va initialiser la configuration locale (import de modules par exemple).

La commande `terraform init` va automatiquement télécharger et installer les binaires des providers nécessaires.

- `terraform plan`

La commande `terraform plan` permet d'afficher le plan d'exécution, qui décrit quelles actions Terraform va prendre pour effectuer les changements réels de l'infrastructure.

Si une valeur est affichée comme `<computed>`, cela veut dire que cette valeur ne sera connue qu'au moment de l'exécution du plan.

- `terraform apply`

La commande `terraform apply` va réellement appliquer les changements tels qu'ils ont été décrits par la commande `terraform plan`.

- `terraform show`

La commande `terraform show` permet d'afficher l'état courant de l'infrastructure.

Une fois que l'infrastructure est gérée via Terraform, il est préférable d'éviter de la modifier manuellement.

Terraform va inscrire des données importantes dans un fichier `terraform.tfstate`. Ce fichier va stocker les ID des ressources créées de façon à savoir quelles ressources sont gérées par Terraform, et lesquelles ne le sont pas. Ce fichier doit donc à son tour être conservé et partagé avec toutes les

personnes devant intervenir sur la configuration.

Ce fichier `terraform.tfstate` contient des données sensibles. Il doit donc être partagé mais de manière sécurisée (des modules existent), éventuellement ailleurs que dans le repo du code de l'infrastructure.

Des ressources déjà existantes peuvent être importées avec la commande `terraform import ressourcetype.name id_existant`.

- `terraform destroy`

Avec l'avènement du cloud, le cycle de vie d'un serveur et notre façon de consommer les ressources ont considérablement changé. Une VM ou une infrastructure doit tout aussi facilement pouvoir être créée que supprimée.

Avec Terraform, une infrastructure complète peut être déployée juste à l'occasion des tests de non régréssion lors de la création d'une nouvelle version logicielle par exemple et être totalement détruite à la fin de ceux-ci pour réduire les coûts d'infrastructure au plus juste.

La commande `terraform destroy` est similaire à la commande `terraform apply`.

Dépendances des ressources

Lorsqu'une ressource dépend d'une autre ressource, la ressource parent peut être appelée comme ceci :

```
# Le VPC de la plateforme
resource "cloudstack_vpc" "default" {
  name         = "our-vpc"
  cidr         = "10.1.0.0/16"
  vpc_offering = "Default VPC offering"
  zone         = "EU-FR-IKDC2-Z4-ADV"
}

# Un réseau en 192.168.1.0/24 attaché à notre VPC
resource "cloudstack_network" "default" {
  name              = "our-network"
  cidr              = "10.1.1.0/24"
  network_offering  = "Isolated VPC tier (100MBps) with SourceNAT and StaticNAT"
  zone              = "EU-FR-IKDC2-Z4-ADV"
  vpc_id            = "${cloudstack_vpc.default.id}"
}
```

Dans l'exemple ci-dessus, un VPC (Virtual Private Cloud) est créé ainsi qu'un réseau privé. Ce réseau privé est rattaché à son VPC en lui fournissant son `id` connu dans terraform en tant que `${cloudstack_vpc.default.id}` où :

- `cloudstack_vpc` est le type de ressource,
- `default` est le nom de la ressource,
- `id` est l'attribut exporté de la ressource.

Les informations de dépendances déterminent l'ordre dans lequel Terraform va créer les ressources.

Les provisionners

Les provisionners permettent d'initialiser les instances une fois qu'elles ont été créées (lancer un playbook ansible par exemple).

Afin de mieux comprendre l'utilité d'un provisionner, étudiez l'exemple ci-dessous :

```
resource "aws_instance" "example" {
  ami           = "ami-b374d5a5"
  instance_type = "t2.micro"

  provisioner "local-exec" {
    command = "echo ${aws_instance.example.public_ip} > ip_address.txt"
  }
}
```

Lors de la création de la nouvelle VM, son adresse IP publique est stockée dans un fichier `ip_address.txt`, mais elle aurait très bien pu être envoyée à la CMDB par exemple.

Le provisioner `local-exec` exécute une commande localement, mais il existe de nombreux autres provisionners : https://www.terraform.io/docs/provisioners/index.html.

Les variables

Une variable est définie comme suit :

```
# variable de type string
variable "region" {
  default = "us-east-1"
}

# variable de type list
# definition implicite
variable "cidrs" { default = [] }

# definition explicite
variable "cidrs" { type = "list" }
```

Pour ensuite être utilisée comme suit :

```
provider "aws" {
  access_key = "${var.access_key}"
  secret_key = "${var.secret_key}"
  region     = "${var.region}"
}
```

Vous pouvez stocker vos variables dans un fichier externe (par exemple `variables.tf`) sachant que tous fichiers ayant pour extension `.tf` du répertoire courant seront chargés.

L'exemple du chapitre précédent peut être repris pour variabiliser la zone de déploiement de nos ressources :

```
# La zone de deploiement cible
variable "zone" {
  default = "EU-FR-IKDC2-Z4-ADV"
}

# Le VPC de la plateforme
resource "cloudstack_vpc" "default" {
  name         = "our-vpc"
  cidr         = "10.1.0.0/16"
  vpc_offering = "Default VPC offering"
  zone         = "${var.zone}"
}

# Un réseau en 192.168.1.0/24 attaché à notre VPC
resource "cloudstack_network" "default" {
  name              = "our-network"
  cidr              = "10.1.1.0/24"
  network_offering  = "Isolated VPC tier (100MBps) with SourceNAT and StaticNAT"
  zone              = "${var.zone}"
  vpc_id            = "${cloudstack_vpc.default.id}"
}
```

Les variables peuvent être également définies depuis la ligne de commande ou un fichier externe `terraform.tfvars` qui sera chargé automatiquement à l'exécution.

Il est d'usage de stocker les variables critiques (api key, mots de passe, etc.) dans un fichier terraform.tfvars et d'exclure ce fichier de votre configuration git.

Voir la documentation pour plus d'informations sur les variables (Maps, etc.) : https://learn.hashicorp.com/terraform/getting-started/variables

TD

Plusieurs possibilités :

- Créer un compte gratuit chez AWS puis :
  - Suivre le module `getting-started` d'hashicorp : https://learn.hashicorp.com/terraform/getting-started/install
  - Créer votre propre infrastructure sur AWS.
- Créer un compte chez Ikoula et gérer une infrastructure CloudStack (https://cloudstack.apache.org/).
- Piloter votre infrastructure VMWare.
- Installer grafana ou un autre logiciel disposant d'un provider sur une de vos VM et utiliser les ressources de ce provider.

Glossaire

BASH
: Bourne Again SHell

BIOS
: Basic Input Output System

CIDR
: Classless Inter-Domain Routing

Daemon
: Disk And Execution MONitor

DHCP
: Dynamic Host Control Protocol

DNS
: Domain Name Service

FIFO
: First In First Out

FQDN
: Fully Qualified Domain Name

GNU
: Gnu is Not Unix

HA
: High Availability (Haute Disponibilité)

HTTP
: HyperText Transfer Protocol

ICP
: Internet Cache Protocol

IFS
: Internal Field Separator

LAN
: Local Area Network

LDIF

LDAP Data Interchange Format

NTP

Network Time Protocol

nsswitch

Name Service Switch

OTP

One Time Password

POSIX

Portable Operating System Interface

POST

Power On Self Test

RDBMS

Relational DataBase Managed System

SGBD-R

Systèmes de Gestion de Base de Données Relationnelles

SHELL

En français "coquille". Á traduire par "interface système".

SMB

Server Message Block

SMTP

Simple Mail Tranfer Protocol

SSH

Secure SHell

SSL

Secure Socket Layer

TLS

Transport Layer Security, un protocole de cryptographie pour sécuriser les communications IP.

TTL

Time To Live

TTY

teletypewriter, qui se traduit téléscripteur. C'est la console physique.

UEFI

Unified Extensible Firmware Interface

Index