



FORMATUX

DevOPS

Antoine Le Morvan, Xavier Sauvignon

Version 2.0 du 16 septembre 2019

Table des matières

Préface	1
Crédits	1
L'histoire de Formatux	2
Licence	3
Comment contribuer au projet ?	3
Antora	5
Gestion des versions	5
1. Généralités DevOPS	6
1.1. Le vocabulaire DEVOPS	6
1.2. Les outils devops	8
2. Premiers pas avec git	10
2.1. Créer un dépôt	10
2.2. Premier ajout de code	14
2.3. Un commit plus complexe	17
2.4. Les commits et les branches	23
2.5. Fusionner des branches	29
2.6. Une fusion de branches échoue	34
3. Gestion de configurations avec Puppet	40
3.1. La gestion de configuration	40
3.2. Puppet	41
3.3. Installation	42
3.4. Hello world	42
3.5. Les modules Puppet	43
3.6. Documentation	44
3.7. Commandes utiles	44
3.8. Cas concrets	45
4. Ansible	48
4.1. Le vocabulaire ansible	49
4.2. Installation sur le serveur de gestion	49
4.3. Utilisation en ligne de commande	50
4.4. Authentification par clef	52
4.5. Utilisation	54
4.6. Les playbooks	56
4.7. La gestion des boucles	59
4.8. Les rôles	60
5. Ansible Niveau 2	62
5.1. Les variables	62

5.2. La gestion des boucles	64
5.3. Les conditions	65
5.4. La gestion des fichiers	67
5.5. Les handlers	69
5.6. Les rôles	70
5.7. Les tâches asynchrones	71
5.8. Connexion à une instance Cloud Amazon ECS	72
6. Ansible Ansistrano	74
6.1. Introduction	74
6.2. Module 1 : Prise en main de la plateforme	75
6.3. Module 2 : Déployer le serveur Web	76
6.4. Module 3 : Déployer le logiciel	78
6.5. Module 4 : Ansistrano	80
6.6. Module 5 : La gestion des branches ou des tags git	86
6.7. Module 6 : Actions entre les étapes de déploiement	89
7. Ordonnanceur centralisé Rundeck	93
7.1. Installation	93
7.2. Configuration	94
7.3. Utiliser RunDeck	94
8. Serveur d'Intégration Continue Jenkins	99
8.1. Installation	99
8.2. Installer Nginx	101
8.3. Configuration de Jenkins	103
8.4. La sécurité et la gestion des utilisateurs	106
8.5. Ajouter une tâche d'automatisation simple	107
8.6. Sources	109
9. DocAsCode : le format AsciiDoc	110
9.1. Introduction	110
9.2. Le format asciidoc	110
9.3. Références	112
10. Infrastructure as Code : Terraform	113
10.1. Introduction	113
10.2. La HCL	114
10.3. Les providers	114
10.4. Les actions	115
10.5. Dépendances des ressources	116
10.6. Les provisionners	117
10.7. Les variables	117
10.8. TD	119

Glossaire	120
Index	121

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 principalement la CentOS, qui est le pendant gratuit de la distribution RedHat, mais la plupart des supports s'appliquent généralement aussi à Ubuntu et Debian. La distribution CentOS est particulièrement adaptée pour un usage sur des serveurs d'entreprises.

Crédits

Ce support de cours a été rédigé sur plusieurs années par de nombreux formateurs.

Les formateurs à l'origine du projet et contributeurs principaux actuels :

- **Antoine Le Morvan** ;
- **Xavier Sauvignon** ;

Notre relecteur principal des premières versions :

- **Patrick Finet** ;

Il a rédigé la partie Git :

- **Carl Chenet**

Ils ont contribué à la rédaction :

- **Nicolas Kovacs** : Apache sous CentOS7 ;
- ...

Enfin, les illustrations de qualité sont dessinées pour formatux par **François Muller** (aka **Founet**).

L'histoire de Formatux

Nous étions (Xavier, Antoine et Patrick) tous les trois formateurs dans la même école de formation pour adulte.

L'idée de fournir aux stagiaires un support en PDF reprenant la totalité des cours dispensés pour leur permettre de réviser et approfondir les points vus en classe pendant la journée nous a rapidement paru être une priorité.

En décembre 2015, nous testions les solutions qui existaient pour rédiger un support d'une telle taille, et nous avons retenu dès le début du projet le format AsciiDoc pour sa simplicité et le générateur AsciiDoctor pour la qualité du support généré, la possibilité de personnaliser le rendu mais surtout pour l'étendue de ses fonctionnalités. Nous avons également testé le markdown, mais avons été plus rapidement limité.



5 ans après le début du projet et après avoir basculé notre site web sous Antora, nous ne regrettons absolument pas le choix technique d'AsciiDoc.

La gestion des sources a été confiée dès l'année suivante à la forge Gitlab de Framagit, ce qui nous permettait de travailler à plusieurs en même temps sur le support, et de faciliter la relecture du support par Patrick. En découvrant la CI de gitlab-ci, nous avons enfin la stack qui nous permettrait d'automatiser totalement la génération du support. Il ne nous manquait plus que la génération d'un site web depuis ces mêmes sources.

Le travail de rédaction étant fortement personnel, sachant que nous risquions d'être muté rapidement dans les années à suivre, nous avons voulu ce support sous Licence Libre, afin qu'un maximum de personnes puissent à la fois contribuer et en profiter, et que notre beau support ne se perde pas.

Même après avoir tous quitté notre organisme de formation, nous avons continué à faire vivre le projet formatux, en faisant évoluer certaines parties, en nous appuyant sur d'autres pour nos formations et en intégrant de nouvelles parties, comme la partie Git de Carl Chenet.

En juillet 2019, nous (Xavier et Antoine, Patrick ayant pris sa retraite informatique) avons décidé de reprendre le développement de Formatux plus activement et d'en faire un peu plus sa promotion. L'organisation complète du support a été revue, en le scindant en 8 dépôts distincts, correspondant à chacune des parties, au support global ainsi qu'au site web. Nous avons voulu notre organisation full devops, afin que la génération de chacune des parties soient totalement automatisées et inter-dépendantes les unes des autres.

Il est difficile aujourd'hui d'évaluer la popularité de notre support. Ce dernier a longtemps été disponible en téléchargement par torrent sur freetorrent (malheureusement aujourd'hui disparu) et en direct depuis le site web. Nous n'avons pas de métriques et nous n'en voulons pas particulièrement. Nous retirons notre satisfaction dans les contacts que nous avons avec nos lecteurs.

Licence

Formatux propose des supports de cours Linux libres de droits à destination des formateurs ou des personnes désireuses d'apprendre à administrer un système Linux en autodidacte.

Les supports de Formatux sont publiés sous licence Creative Commons-BY-SA et sous licence Art Libre. Vous êtes ainsi libre de copier, de diffuser et de transformer librement les œuvres dans le 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 formatux.fr :

- <https://www.formatux.fr/>

Les sources de nos supports sont hébergées chez Framasoft sur leur forge Framagit. Vous y trouverez les dépôts des codes sources à l'origine de la version de ce document :

- <https://framagit.org/formatux/>

A partir des sources, vous pouvez générer votre support de formation personnalisé. Nous vous recommandons le logiciel AsciiDocFX téléchargeable ici : <http://asciidocfx.com/> ou l'éditeur Atom avec les plugins AsciiDoc.

Comment contribuer au projet ?

Si vous voulez participer à la rédaction des supports formatux, **forkez-nous sur framagit.org**.

Vous pourrez ensuite apporter vos modifications, compiler votre support personnalisé et nous proposer vos modifications.

Vous êtes les bienvenus pour :

- Compléter le document avec un nouveau chapitre,
- Corriger ou compléter les chapitres existants,
- Relire le document et corriger l'orthographe, la mise en forme,

- Promouvoir le projet

De votre côté

1. Créer un compte sur <https://framagit.org>,
2. Créer un fork du projet que vous voulez modifier parmi la liste des projets du groupe : [Créer le fork](#),
3. Créer une branche nommée develop/[Description],
 - Où [Description] est une description très courte de ce qui va être fait.
4. Faire des commits dans votre branche,
5. Pusher la branche sur votre fork,
6. Demander une merge request.



Si vous n'êtes pas un grand utilisateur de git, ce n'est pas grave. Vous pouvez toujours lire la partie Git de notre support puis nous contacter. Nous vous guiderons ensuite pour vos premiers pas.

Cette remarque est également vraie pour le format AsciiDoc.

Essayer de conserver le même ton qui est déjà employé dans le reste du support (pas de 'je' ni de smileys par exemple).

La suite se passe de notre côté

1. Quelqu'un relira votre travail,
 - Essayez de rendre ce travail plus facile en organisant vos commits.
2. S'il y a des remarques sur le travail, le relecteur fera des commentaires sur la merge request,
3. Si la merge request lui semble correcte il peut merger votre travail avec la branche **develop**.

Corrections suite à une relecture

La relecture de la merge request peut vous amener à faire des corrections. Vous pouvez faire ces corrections dans votre branche, ce qui aura pour effet de les ajouter à la merge request.

Comment compiler mon support formatux ?

Après avoir forké notre projet ou l'avoir cloné (`git clone https://framagit.org/group/formatux-PARTIEXX.git`), déplacez-vous dans le dossier formatux-PARTIEXX nouvellement créé.

Vous avez ensuite plusieurs possibilités :

- Vous utilisez le logiciel AsciiDocFX ou Atom avec ses plugins AsciiDoc (recommandé sous Windows) : lancez le logiciel, ouvrez le fichier `.adoc` désiré, et cliquez sur le bouton **PDF**

(AsciidoctorFX) ou regardez la prévisualisation (Atom).

- Vous êtes sous Linux et vous avez déjà installé le paquet asciidoctor : exécutez la commande `asciidoctor-pdf -t -D public -o support.pdf support.adoc`.

Comment faire un don

Vous pouvez faire un don par paypal pour soutenir nos efforts. Ces dons serviront à payer notre nom de domaine et notre hébergement. Nous pourrions également reverser une partie de ces dons à Framasoft, qui héberge gracieusement nos repos et met à disposition un runner qui compile nos supports et notre site web. Enfin, une petite bière entre nous peut également être au programme.

Accès à la cagnotte paypal : <https://www.paypal.com/pools/c/8hlM1Affp1>.

Nous contacter

Nous sommes disponibles sur gitter pour échanger autour de formatux, de Linux et de la formation : <https://gitter.im/formatux-fr/formatux>.

Antora

Pour la génération de notre site web, nous utilisons Antora. Antora nous permet, depuis les mêmes sources, de pouvoir générer le support en PDF et le site web statique en HTML. Le développement d'Antora est sponsorisé par OpenDevise Inc.

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
2.0	Septembre 2019	Passage à antora

Chapitre 1. Généralités DevOPS

Découvrir la philosophie **devops**.

Objectifs pédagogiques

🔗 **devops, automatisation, gestion de configuration, intégration continue, déploiement continu, 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.

1.1. 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.

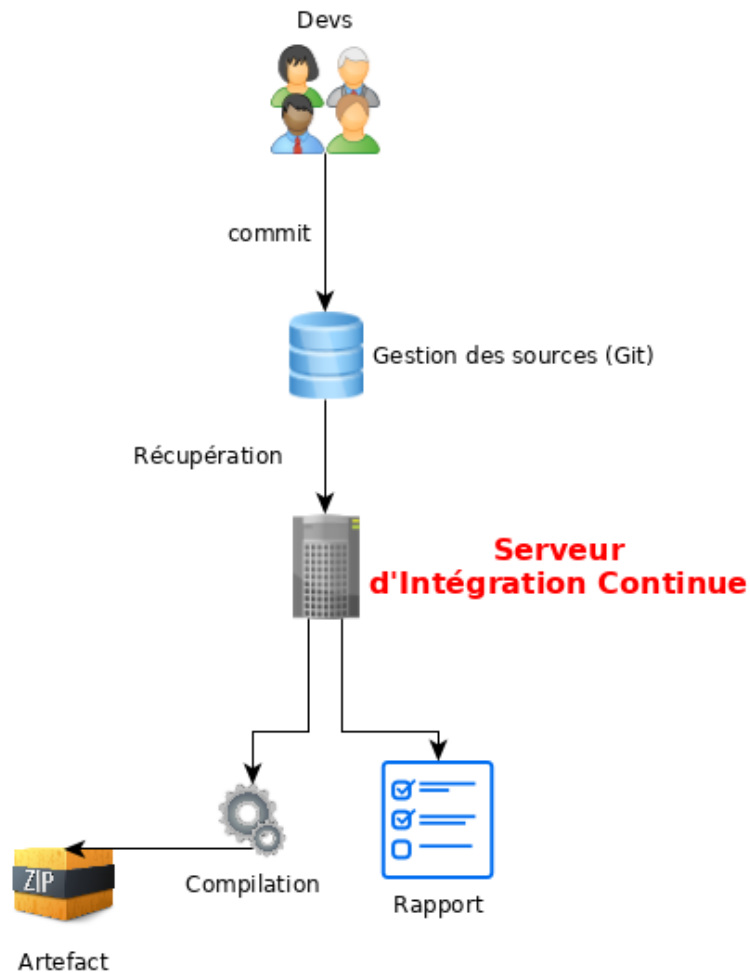


Figure 1. Schéma de fonctionnement de l'intégration continue

- **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 quelle soit appliquée une ou plusieurs fois. Les outils de gestion de configuration sont généralement idempotents.
- **orchestration** : processus automatique de gestion d'un système informatique.
- **provisionnement** : processus d'allocation automatique de ressources.

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

Les langages impératifs contre les langages 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 ceux utilisés par Ansible, Terraform, ou Puppet, quant à eux, ne spécifient pas les étapes à réaliser mais l'état à obtenir.

Parce qu'Ansible, Terraform 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.

1.2. Les outils devops

- Outils de gestion de configuration
 - Puppet
 - Ansible
 - Saltstack
 - Chef
- Intégration continue
 - Jenkins
 - Gitlab-ci
- Orchestration des tâches
 - Rundeck
 - Ansible Tower
- Infrastructure As Code
 - Terraform
- Docs as Code

-
- AsciiDoctor
 - Markdown
 - ReStructured Text

Chapitre 2. Premiers pas avec git

Ces articles ont été initialement rédigés par [Carl Chenet](#) et repris par formatux avec son aimable autorisation.

Git reste un programme incompris et craint par beaucoup alors qu'il est aujourd'hui indistinctement utilisé par les développeurs et les sysadmins. Afin de démystifier ce formidable outil, je vous propose une série d'articles à l'approche très concrète pour débiter avec Git et l'utiliser efficacement ensuite.



2.1. Créer un dépôt

La première partie de cette série va se concentrer sur la création d'un dépôt Git. Afin de refléter l'usage actuel qui allie le plus souvent un dépôt Git local et un dépôt distant, nous nous appuierons sur [Gitlab.com](#), excellente forge logicielle en ligne basée sur le logiciel libre [Gitlab](#).



Vous pouvez également utiliser l'instance gitlab de [framagit](#) qui héberge notamment les sources de Framatux.

Qu'est-ce qu'un dépôt de sources ?

Le dépôt de sources est très concrètement un répertoire sur votre système, contenant lu-même un répertoire caché nommé `.git` à sa racine. Vous y stockerez votre code source et il enregistrera les modifications apportées au code dont il a la charge.

Pour prendre directement de bonnes habitudes, nous allons créer un nouveau projet à partir de notre compte Gitlab.com. Un projet Gitlab offre un dépôt distant et une suite d'outils autour. Dans un premier temps nous ne nous intéresserons qu'au dépôt. Le but est, dans un premier temps, de montrer le lien entre le dépôt distant et le dépôt local.

Pourquoi un dépôt distant ?

Tout simplement dans un premier temps pour sauver votre code sur un autre ordinateur que le vôtre, et dans un second temps de permettre le travail collaboratif.

Créer un projet sur Gitlab.com

Nous commençons par créer un compte sur Gitlab.com.



GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- [Explore projects on GitLab.com](#) (no login needed)
- [More information about GitLab.com](#)
- [GitLab.com Support Forum](#)
- [GitLab Homepage](#)

By signing up for and by signing in to this service you accept our:

- [Privacy policy](#)
- [GitLab.com Terms](#).

Sign in	Register
Full name	
<input type="text"/>	
Username	
<input type="text"/>	
Email	
<input type="text"/>	
Email confirmation	
<input type="text"/>	

Une fois connecté, nous créons maintenant un nouveau projet.

The screenshot shows the GitLab.com web interface. At the top, there's a dark navigation bar with the GitLab logo, 'Projects' dropdown, 'Groups', 'More', a search bar, and user avatars. Below this is a blue banner stating 'Project 'Carl Chenet / toto' is in the process of being deleted.' The main section is titled 'Projects'. On the right side of this section, a green button labeled 'New project' is circled in red. Below the 'Projects' title, there are tabs for 'Your projects 61', 'Starred projects 11', and 'Explore projects'. There's also a 'Filter by name...' input field and a 'Last updated' dropdown menu. Under the 'All' tab, a project by 'Carl Chenet / feed2toot' is listed, with a 'Maintainer' badge and a star icon indicating 52 stars. The project description says 'Feed2toot automatically parses rss feeds, identifies new posts and posts them on the Mast...' and it was 'Updated 49 minutes ago'.

Nous lui donnons également un nom. Ce projet n'ayant pas (encore) comme but de devenir public, nous restons en dépôt privé, ce qui signifie que l'accès au dépôt sera restreint par un identifiant et un mot de passe.

Blank project

Create from template

Import project

CI/CD for external repo

Project name

toto

Project URL

https://gitlab.com/chaica

Project slug

toto

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Description format

Visibility Level

☒ Private
Project access must be granted explicitly to each user.

☐ Internal
The project can be accessed by any logged in user.

☐ Public
The project can be accessed without any authentication.

☐ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

Nous en avons fini avec Gitlab.com. De retour à Git et à la ligne de commande.

Cloner un dépôt distant

Sur notre poste, nous allons commencer par cloner le dépôt distant depuis Gitlab.com vers notre poste local :

```
$ git clone https://gitlab.com/chaica/toto.git
Cloning into 'toto'...
warning: You appear to have cloned an empty repository.
$ cd toto
$ ls -a
.  ..  .git
```

Comment se souvenir d'où vient ce code ? La commande `git remote` va nous permettre de voir le lien entre notre dépôt local et le dépôt distant de Gitlab.com :


```
$ git remote -v
origin https://gitlab.com/chaica/toto (fetch)
origin https://gitlab.com/chaica/toto (push)
```

Nous détaillerons plus tard, l'important est de voir qu'il y a bien un lien entre le dépôt local et le dépôt distant.

S'identifier

Le dépôt Git dans lequel vous travaillez va bientôt chercher à identifier la personne qui procède à des modifications. Pour cela nous allons définir notre identité au niveau local – c'est-à-dire de ce dépôt – avec les commandes suivantes :

```
$ git config --local user.name "Carl Chenet"
$ git config --local user.email "chaica@ohmytux.com"
```

Git crée en fait ici un fichier `.git/config` contenant les informations fournies.

Vous pouvez également définir votre identité au niveau global, pour ré-utiliser cette configuration pour tous les dépôts que vous créerez avec cet utilisateur du système :

```
$ git config --global user.name "Carl Chenet"
$ git config --global user.email "chaica@ohmytux.com"
```

Git crée en fait ici un fichier `~/.gitconfig` contenant les informations fournies.

Saisir le mot de passe le moins possible

Lorsque nous avons cloné notre dépôt plus haut dans l'article, vous avez dû saisir un mot de passe. J'imagine que vous avez choisi un mot de passe à 24 caractères pour protéger votre compte Gitlab.com, qui est donc également le mot de passe de votre dépôt distant. Dans les prochains articles, nous allons beaucoup interagir avec le dépôt distant et il serait assez pénible d'avoir à le saisir régulièrement. Pour remédier à cela nous allons immédiatement passer la commande suivante :

```
$ git config --local credential.helper cache
```

Le dernier argument indique le type de stockage du mot de passe. Ici il s'agit uniquement d'un cache d'une durée de vie égale à 15 minutes. Si vous souhaitez stocker définitivement votre mot de passe, vous pouvez utiliser la commande suivante :

```
$ git config credential.helper store
```

Vos mots de passe seront stockés par défaut dans `~/.git-credentials`. Attention, les mots de passe sont sauves en clair dans ce fichier.

Conclusion

Pour une première approche de Git, nous avons appris à créer un projet sur Gitlab.com, à rapatrier localement le dépôt créé puis à le configurer pour l'utiliser le plus simplement possible. Nous verrons les premières utilisations de ce dépôt dans une prochaine partie.

2.2. Premier ajout de code

Après avoir vu comment débiter avec Git, nous allons détailler comment ajouter pour la première fois du code dans un dépôt de code source et les notions fondamentales de Git qui accompagnent ces opérations.

État initial du dépôt

Dans un premier temps, nous allons nous positionner dans le dépôt que nous avons créé durant la première partie sur notre poste de travail local et vérifier l'état courant de notre dépôt :

```
$ cd ~/toto
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Plusieurs notions importantes dans le résultat de cette commande :

- Le message **On branch master** vous signale la branche dans laquelle vous êtes positionné.
- Mais qu'est-ce qu'une branche ? Retenons ici essentiellement que nous travaillons dans un état particulier du dépôt appelé **branche** et qu'il s'agit de celle par défaut, nommée **master**. Nous serons beaucoup plus précis sur cette notion dans une prochaine partie.
- Le message **No commits yet** est assez explicite et indique qu'aucun **commit** n'a encore été enregistré dans le dépôt, ce qui est cohérent.
- Mais qu'est-ce qu'un **commit** ? Un **commit** est l'état du code enregistré dans votre dépôt à un moment T, un instantané de tous les éléments dans votre dépôt au moment où vous effectuez le commit.

Allons donc maintenant ajouter un fichier dans la branche master de notre dépôt :

```
$ touch README.md
$ git add README.md
```

Créons maintenant un fichier vide nommé **README.md**. Nous utilisons ensuite la commande **git add README.md** afin d'ajouter le fichier sur la branche. Après cela, le nouvel état du dépôt est le suivant :

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md
```

Nous observons que nous sommes toujours sur la branche **master** et qu'aucun **commit** n'a toujours été effectué sur cette branche. La section suivante indique par contre qu'un changement est désormais pris en compte par Git et qu'il s'agit d'un nouveau fichier nommé **README.md**.

Notion importante ici à retenir : la commande **git add** nous a permis de faire passer une modification – ici l'ajout d'un fichier – de l'état où ce fichier n'était pas pris en compte par Git vers un espace de travail, appelé ici **stage**, ce qui signifie donc que ce fichier est désormais surveillé par Git.

Premier commit

Premier grand événement de la vie de notre dépôt, nous allons maintenant effectuer notre premier **commit** avec la commande suivante :

```
$ git commit README.md -m "add README.md"
[master (root-commit) 505ace4] add README.md
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

La command **git commit** nous permet de spécifier un fichier à prendre en compte pour le commit en question. Ici c'était facile, nous n'en avons qu'un. L'option **-m** accompagnée d'un texte permet de spécifier un message explicatif à associer à ce commit. Git nous retourne différentes informations sur le commit effectué puis nous vérifions de nouveau l'état du dépôt :

```
$ git status
On branch master
nothing to commit, working tree clean
```

Nous n'apprenons pas grand-chose. On peut remarquer que le message **No commit yet** a disparu. Nous allons passer une commande spécifique pour consulter l'historique des commits :

```
$ git log
commit 5c1b4e9826a147aa1e16625bf698b4d7af5eca9b
Author: Carl Chenet <chaica@ohmytux.com>
Date: Mon Apr 29 22:12:08 2019 +0200

add README.md
```

La commande **git log** nous apprend l'identifiant, l'auteur et la date du commit, suivi par le message explicatif dudit commit.

Pousser son commit vers le dépôt distant

Dans la première partie de cette série, nous avons créé un dépôt distant à partir duquel avait été cloné le dépôt local dans lequel nous venons de travailler. Il s'agit maintenant de synchroniser le dépôt distant avec le dépôt local. Pour cela, nous utilisons la commande suivante :

```
$ git push --set-upstream origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 220 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.com/chaica/toto
* [new branch] master -> master
Branch master set up to track remote branch master from origin.
```

La commande **git push** nous permet de pousser nos modifications d'un dépôt local vers un dépôt distant. L'option **--set-upstream** permet d'indiquer que notre branche courante dans le dépôt local (désigné par le terme **origin** et qui est donc ici nommée **master**) sera poussée vers la branche du dépôt distant nommée elle aussi **master**. Cette option n'est obligatoire qu'à votre premier push.

Git nous indique ici **[new branch]** car, pour rappel, nous avons cloné un dépôt vide. Il crée donc la branche du dépôt distant nommé **master**. Refaisons appel à la commande **git remote** que nous avons déjà utilisée dans la première partie. En effet elle va nous permettre de mieux appréhender le rapport entre la branche locale et la branche distante :

```
$ git remote -v
origin https://gitlab.com/chaica/toto (fetch)
origin https://gitlab.com/chaica/toto (push)
```

Nous voyons sur la seconde ligne qu'en effet l'origine pour la branche en cours, ici l'url <https://gitlab.com/chaica/toto>, pour l'action **push** est bien notre dépôt distant.

Nous pouvons maintenant utiliser une version beaucoup plus simple de la commande `git push` se limitant à deux mots :

```
$ git push
Everything up-to-date
```

Le message de Git est clair, notre dépôt local est à jour par rapport au dépôt distant. Nos récentes modifications, représentées au niveau de Git par le commit que nous venons de créer, ont été poussées vers le dépôt distant, assurant la redondance des données et permettant – nous le verrons plus tard dans quelques parties – le travail collaboratif.

Conclusion

Cette deuxième partie sur comment débiter avec Git nous a permis de réaliser les toutes premières opérations d'ajouts de code sur notre dépôt local et de s'assurer de sa synchronisation avec le dépôt distant. Les notions de branche et de commit ont également été abordées pour la première fois. Dans une prochaine partie nous présenterons comment gérer un commit plus complexe.

2.3. Un commit plus complexe

Après avoir réalisé notre premier commit, nous continuons en nous intéressant aux différentes opérations possibles pour réaliser un commit plus complexe que le simple ajout d'un fichier.

Objectif

Pour bien débiter avec Git, nous avons vu dans la partie précédente comment réaliser un commit très simple, consistant à enregistrer un seul fichier. Nous allons aller plus loin pour coller davantage à la complexité du monde réel en constituant un nouveau commit avec plusieurs modifications différentes issues de plusieurs fichiers.

État initial du dépôt

Nous allons commencer avec un dépôt Git contenant un seul fichier texte nommé `README` qui est assez long.

```
$ cd commit-complexe
$ ls
README
$ wc -l README
19 README
```

Premiers changements

Nous allons ajouter sur la première ligne la phrase “do not forget to read the file foo!” et sur la

dernière ligne la phrase “do not forget to read the file bar”.

Voyons maintenant les modifications par rapport au commit précédent :

```
$ cd commit-complexe
$ git diff
diff --git a/README b/README
index d012f47..737bc05 100644
--- a/README
+++ b/README
@@ -1,5 +1,7 @@
this is a README

+do not forget to read the file foo!
+
Lots of interesting stuff here

Let's work
@@ -17,3 +19,5 @@ These criteria eliminated every then-extant version-control system,
so immediate
The development of Git began on 3 April 2005.[16] Torvalds announced the project on 6
April;[17] it became self-hosting as of 7 April.[16] The first merge of multiple
branches took place on 18 April.[18] Torvalds achieved his performance goals; on 29
April, the nascent Git was benchmarked recording patches to the Linux kernel tree at
the rate of 6.7 patches per second.[19] On 16 June Git managed the kernel 2.6.12
release.[20]

Torvalds turned over maintenance on 26 July 2005 to Junio Hamano, a major contributor
to the project.[21] Hamano was responsible for the 1.0 release on 21 December 2005 and
remains the project's maintainer.[22]
+
+To finish, do not forget to read file bar!
```

Git nous indique les changements intervenus avec des + devant les ajouts. Nous avons donc rajouté deux lignes et deux sauts de lignes.

Choix de ce qu'on souhaite mettre dans son commit : l'index

Bien, il nous reste maintenant à écrire les nouveaux fichiers **foo** et **bar**. Mais il est 19h, je suis fatigué et je ne vais avoir le temps que d'écrire le premier fichier **foo**. Nous procédons comme suit pour écrire et ajouter le fichier **foo** :

```
$ echo "very interesting stuff" > foo
$ cat foo
very interesting stuff
```

Une fois créé, nous indiquons à Git que nous souhaitons que ce nouveau fichier soit pris en compte au prochain commit. On appelle cette opération indexer un fichier :

```
$ git add foo
```

Une furieuse envie d'enregistrer le travail en cours en faisant un `git commit -a` est notre premier réflexe, mais le fichier `README` va donc évoquer un fichier `bar` qui n'existe pas encore dans le dépôt. Ce n'est pas propre, surtout si un collègue relit immédiatement mon travail.

Dans notre situation, la solution est de choisir ce que l'on veut ajouter au commit, c'est à dire le fichier `foo` (ce que nous venons de faire) et seulement une partie des modifications du fichier `README`. C'est possible avec l'option `--patch` ou `-p` de `git add` qui va nous permettre d'indexer seulement les modifications qui nous intéressent pour préparer le prochain commit.

```
$ git add -p README
```

La commande va identifier vos différentes modifications du fichier en question et vous demander lesquelles vous souhaitez indexer pour le prochain commit.

```
@@ -1,5 +1,7 @@  
this is a README  
  
+do not forget to read the file foo!  
+  
Lots of interesting stuff here  
  
Let's work  
Stage this hunk [y,n,q,a,d,j,J,g,/,e,?]? y
```

Cette commande nous présente notre première modification dans le fichier `README`. Un `hunk` est ici donc une modification unitaire identifiée par Git. Le contenu ajouté apparaît avec un symbole `+` en tout début de ligne. Nous acceptons d'indexer la première modification en appuyant sur `y` (yes).

```
+  
+To finish, do not forget to read file bar!  
Stage this hunk [y,n,q,a,d,K,g,/,e,?]? n
```

Git nous affiche ensuite la seconde modification identifiée dans le fichier. Nous refusons la seconde en appuyant sur `n` (no), que nous ajouterons demain quand nous aurons écrit le fameux fichier `bar` dans un futur commit, parce qu'il est 19h et qu'à chaque jour suffit sa peine.

L'index avant le commit

Bien, après cette sélection, où en sommes-nous ? Un **git status** va nous aider.

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README
    new file:   foo

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   README
```

La commande est très claire si on la lit rigoureusement : sur la branche master, les modifications suivantes sont prêtes à être committées : le fichier **README** a été modifié et un nouveau fichier **foo** a été créé.

Git nous indique également qu'un changement est survenu mais qu'il n'a pas été indexé (sous la phrase Changes not staged for commit) : c'est le fameux changement que nous souhaitons faire demain.

Le commit lui-même

Il est grand temps de valider nos modifications et de créer un commit. Il ne faut ici surtout pas utiliser l'option **-a** de **git commit** sous peine de valider indistinctement toutes les modifications présentes dans le dépôt, ce que nous ne voulons absolument pas faire.

On va vérifier puis valider notre commit grâce à la commande suivante :

```
$ git commit -v
```

Un éditeur de texte s'ouvre et nous affiche la sortie suivante :


```

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   modified:   README
#   new file:   foo
#
# Changes not staged for commit:
#   modified:   README
#
# ----- >8 -----
# Do not modify or remove the line above.
# Everything below it will be ignored.
diff --git a/README b/README
index d012f47..6c50e6b 100644
--- a/README
+++ b/README
@@ -1,5 +1,7 @@
this is a README

+do not forget to read the file foo!
+
Lots of interesting stuff here

Let's work
diff --git a/foo b/foo
new file mode 100644
index 0000000..7c50c6a
--- /dev/null
+++ b/foo
@@ -0,0 +1 @@
+very interesting stuff

```

Nous retrouvons ici le résultat du précédent `git status` et le détail des modifications qui vont être validées. Nous remarquons que seule la mention du fichier `foo` dans le `README` et la création de ce fichier `foo` y figurent. Nous avons donc réussi notre commit.

Si ça n’était pas le cas, vous pouvez quitter votre éditeur de texte sans enregistrer, cela annulera le commit. Si tout est bon, entrez simplement un message de commit pertinent comme “mention foo in README and add the foo file” et sauvegardez.

Le résultat de la commande est le suivant :

```
$ git commit -v
[master 077f1f6] mention foo in README and add the foo file
2 files changed, 3 insertions(+)
create mode 100644 foo
```

Nous voyons notre message de validation et également que deux fichiers ont été modifiés.

État de notre dépôt après ce commit

Dans quel état est notre dépôt après ce commit ? Encore une fois l'analyse de la commande **git status** va nous aider.

```
$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified: README

no changes added to commit (use "git add" and/or "git commit -a")
```

En effet nous avons la seconde modification du fichier README que nous avons faite qui n'a pas encore été enregistrée dans aucun commit. Nous n'avons pas indexé cette modification. Elle demeure donc dans notre dépôt en attente d'un prochain traitement.

Un **git diff** nous montre cette modification :

```
$ git diff
diff --git a/README b/README
index 6c50e6b..737bc05 100644
+
+To finish, do not forget to read file bar!
```

Il s'agit bien de la modification que nous ne souhaitons pas encore enregistrer.



J'ai simplifié l'affichage de la commande précédente pour aller à l'essentiel.

Sauvegarde du travail

Nous avons un beau commit, mais il n'existe pour l'instant que sur notre ordinateur personnel. Un malheureux accident ou un vol est si vite arrivé, il est indispensable de sauver notre travail sur notre dépôt Git distant (voir la création du dépôt distant dans la partie 1). Nous procédons avec un simple **git push** vers notre dépôt distant:

```
$ git push
```

Conclusion

Bien débiter avec Git nécessite de comprendre ce que l'on fait et pour cela nous abordons les notions fondamentales une par une.

La notion essentielle abordée aujourd'hui est l'ajout de modifications à l'index. L'index constitue l'ensemble des modifications à prendre en compte dans le prochain commit. Nos modifications ont donc pour l'instant 3 statuts possible dans Git : non-indexées, indexées ou enregistrées dans un commit. À chaque statut correspond une palette d'opérations possibles et Git vous tient en permanence au courant du statut de vos différentes modifications en cours, en vous proposant souvent des actions possibles.

Nous aborderons bientôt dans une prochaine partie une autre notion fondamentale de Git : les branches.

2.4. Les commits et les branches

Dans cette série consacrée à l'apprentissage pratique de Git à travers des exemples, après avoir vu ce qu'est un commit, nous étudierons comment s'organisent les commits et comment passer de l'un à l'autre.

Objectif

Comme nous l'avons vu à la partie 2 et à la partie 3, Git enregistre les modifications qui surviennent au code dans le dépôt sous forme de commits.

Au fil des commits, nous construisons donc un historique des nos modifications. Git nous permet de naviguer entre ces modifications et donc de retrouver les états antérieurs des sources dans notre dépôt. Nous allons aujourd'hui détailler les possibilités offertes par cette organisation des commits.

État initial du dépôt

Nous nous positionnons dans un dépôt Git contenant actuellement deux fichiers.

```
$ cd historique-commit  
$ ls  
file1 file2
```

Le dépôt Git a connu 4 commits, comme nous l'indique la commande **git log**.

```
$ git log
commit ab63aad1cfa5dd4f33eae1b9f6baf472ec19f2ee (HEAD -> master)
Author: Carl Chenet <chaica@ohmytux.com>
Date: Tue May 28 20:46:53 2019 +0200

    adding a line into file 2

commit 7b6882a5148bb6a2cd240dac4d339f45c1c51738
Author: Carl Chenet <chaica@ohmytux.com>
Date: Tue May 28 20:46:14 2019 +0200

    add a second file

commit ce9804dee8a2eac55490f3aee189a3c67865481c
Author: Carl Chenet <chaica@ohmytux.com>
Date: Tue May 28 20:45:21 2019 +0200

    adding a line in file 1

commit 667b2590fedd4673cfa4e219823c51768eeaf47b
Author: Carl Chenet <chaica@ohmytux.com>
Date: Tue May 28 20:44:30 2019 +0200

    first commit
```

La commande **git status** nous précise quant à elle qu'aucun travail n'est en cours.

```
$ git status
On branch master
nothing to commit, working tree clean
```

Affichons le dernier fichier modifié pour la suite de l'article.

```
$ cat file2
this is the number 2 file

adding a line into file 2
```

Retrouver un état antérieur

Nous allons maintenant tenter de retrouver un état antérieur de notre dépôt, à savoir l'état de notre dépôt au précédent commit.

La commande **git checkout** va nous permettre de revenir à l'état de notre dépôt à un certain commit. Nous pouvons utiliser pour ça un nombre de commits antérieurs, par exemple juste 1

commit avant :

```
$ git checkout HEAD~1
```

Nous pouvons également utiliser l'identifiant du commit.

```
$ git checkout 7b6882a5148bb6a2cd240dac4d339f45c1c51738
Note: checking out '7b6882a5148bb6a2cd240dac4d339f45c1c51738'.
```

You are **in** **'detached HEAD'** state. You can look around, make experimental changes and commit them, and you can discard any commits you make **in** this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may **do** so (now or later) by using **-b** with the checkout **command** again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 7b6882a add a second file
```

La sortie de Git est un peu difficile à comprendre tout de suite, mais le fait est que nous sommes revenus à l'état du dépôt à l'avant-dernier commit.

Affichons le dernier fichier que nous avons modifié.

```
$ cat file2
this is the number 2 file
```

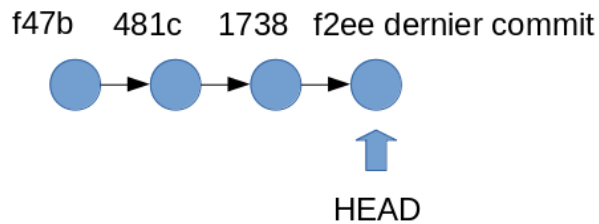
Son contenu a bien changé, nous sommes donc bien revenus en arrière dans l'histoire des modifications de notre dépôt.

Le pointeur HEAD

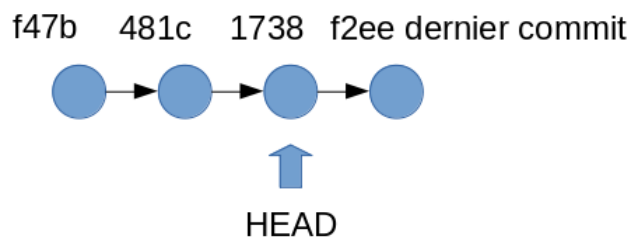
Un composant important de la commande **git** précédente reste assez obscure : que signifie **HEAD** ? Et pourquoi **~1** ?

Il s'agit tout simplement d'un pointeur de position parmi les commits de notre dépôt. Un pointeur est ici un repère logique, un petit drapeau au sein de Git, qui indique un commit et que l'on peut déplacer pour indiquer un autre commit.

Un schéma va nous aider à comprendre. Nous identifions les commits par les 4 derniers caractères de leurs identifiants.



Avant notre checkout, **HEAD** pointait sur le dernier commit réalisé. Après le **git checkout HEAD~1**, nous avons positionné **HEAD** sur l'avant-dernier commit.



Nous entrons dans un mode spécial de Git (**detached head** ou tête détachée), qui nous permet de retrouver les données du dépôt telles qu'elles étaient au moment de ce commit. À partir de cet état du dépôt, nous pourrions évoluer vers un autre état sans modifier les commits déjà existants.

Différences entre deux commits

Nous allons maintenant observer les différences entre le commit sur lequel nous sommes positionnés et le commit le plus avancé que nous avons écrit, à l'aide de la commande **git diff**.

```
$ git diff HEAD master
diff --git a/file2 b/file2
index a21d8c9..040c455 100644
--- a/file2
+++ b/file2
@@ -1,3 @@
this is the number 2 file
+
+adding a line into file 2
```

Nous voyons bien la ligne ajoutée au fichier **file2** lors du dernier commit.

Remarquons que nous avons utilisé dans notre commande **master**, avec **HEAD**. Ici **HEAD** point sur l'avant-dernier commit de notre liste. Nous voyons les différences entre l'avant-dernier et le dernier commit. Or le dernier argument de notre ligne de commande était **master**. Il s'agit donc aussi, comme **HEAD**, d'un pointeur, mais sur le dernier commit réalisé. Nous y reviendrons.

Cette commande **git diff** marche bien sûr avec n'importe quel identifiant de commit, par exemple voici la différence entre le premier et le second commit, en utilisant ici leur identifiant unique.

```
$ git diff 667b2590fedd4673cfa4e219823c51768eeaf47b
ce9804dee8a2eac55490f3aee189a3c67865481c
diff --git a/file1 b/file1
index 9dd524a..2788b18 100644
--- a/file1
+++ b/file1
@@ -1,3 @@
this is the number 1 file
+
+adding a line in file 1
```

Les différences entre le premier et le second commit apparaissent bien.

Écrire une suite différente : une nouvelle branche

Nous sommes donc positionnés sur l'avant-dernier commit. Nous nous apercevons que nous aimerions continuer avec un contenu différent que celui enregistré dans le dernier commit, sans toutefois effacer ce dernier commit. Pour résumer nous voulons créer un embranchement dans l'histoire de nos commits pour créer une suite différente. Nous allons donc créer notre première branche.

Pour cela il suffit de relire le dernier message affichée lors de notre commande `git checkout HEAD~1` :

```
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
```

```
git checkout -b <new-branch-name>
```

Nous allons donc passer la commande suivante afin de créer une nouvelle branche dans laquelle nous écrirons la nouvelle suite des modifications que nous souhaitons.

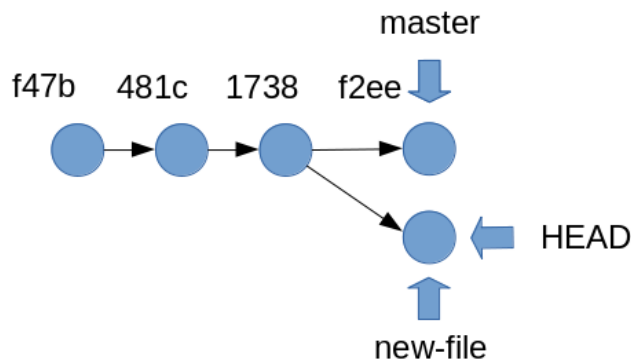
```
$ git checkout -b new-file
$ git status
On branch new-file
nothing to commit, working tree clean
```

Remarquons la première ligne `On branch new-file` alors que jusqu'ici nous avions `On branch master`. Nous avons donc bien créé une nouvelle branche nommée `new-file`.

Nous créons maintenant un nouveau commit contenant un nouveau fichier et l'ajoutons au dépôt.

```
$ echo "An unexpected new file" > file3
$ git add file3
$ git commit file3 -m "create an unexpected file"
[new-file a2e05c3] create an unexpected file
1 file changed, 1 insertion(+)
create mode 100644 file3
```

Où en sommes-nous ? Un schéma vaut mieux qu'un long discours.



Ce schéma nous apprend beaucoup de choses :

- notre première série de commits finit par un commit **f2ee** sur lequel un pointeur nommé **master** est positionné. Il s'agit de la branche **master**
- De la même façon, la branche **new-file** pointe sur notre dernier commit.
- Le pointeur **HEAD** indique l'état du dépôt sur lequel on travaille.

Une branche est donc définie par une série de commits et un pointeur du nom de cette branche sur le dernier commit de cette branche.

Conclusion

Arrêtons-nous là pour l'instant. Nous avons vu une notion fondamentale, à savoir ce qu'est réellement une branche Git et les principes sous-jacents à une branche, le lien entre les commits et les pointeurs. Il était malheureusement difficile de parler des branches précisément (ce que nous avons fait dans notre première partie) sans toutes ces notions.

Dans un dépôt Git, l'unité est le **commit**, qui est un ensemble de modifications survenus sur le code dans ce dépôt. Un **commit** et ses prédécesseurs représentent une **branche**. On positionne sur certains **commits** des **pointeurs**, ayant chacun un rôle distinct :

- Le pointeur nommé **master** pointe sur le dernier commit de la branche **master**.
- Le pointeur **new-file** pointe sur le dernier commit de la branche éponyme.
- Un pointeur spécial nommé **HEAD** indique en permanence l'état du dépôt au dernier commit pointé par le pointeur **HEAD**.

Nous verrons dans une prochaine partie comment les branches interagissent entre elles et comment les utiliser pour multiplier les différentes versions d'un travail en cours.

2.5. Fusionner des branches

Nous avons vu ce que sont les branches Git. Nous allons maintenant présenter pourquoi fusionner des branches et comment s'y prendre.

Mise en place

Pour cette partie nous créons un dépôt distant sur notre Gitlab (comme expliqué dans la partie 1), puis nous le clonons sur notre machine locale.

```
$ git clone https://gitlab.com/chaica/merge-branches.git
Cloning into 'merge-branches'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
$ cd merge-branches/
$ ls
README.md
```

Notre dépôt initial contient simplement un fichier **README.md**, créé par Gitlab, contenant un titre en langage **Markdown**

```
$ cat README.md
# merge-branch
```

Nous allons maintenant ajouter un sous-titre à ce fichier pour un prochain exemple.

```
$ echo -e "\n## sous-titre" >> README.md
$ git commit README.md -m "add first subtitle"
[master 2059805] add first subtitle
1 file changed, 2 insertions(+)
```

À ce niveau, nous avons donc deux commits qui constituent notre branche master, comme nous l'indique la commande **git log**.

```
$ git log
commit 20598053fb2c3e55f95e2521dfa804739abd7d8a
Author: Carl Chenet chaica@ohmytux.com
Date:   Fri Jun 21 10:22:00 2019 +0200

    add first subtitle

commit 11cb68a24bed5236972138a1211d189adb4512a8 (origin/master, origin/HEAD)
Author: Carl Chenet chaica@ohmytux.com
Date:   Fri Jun 21 08:18:56 2019 +0000

    Initial commit
```

Mise en place un peu longue, mais qui nous a permis de réviser quelques commandes fondamentales. Nous entrons dans le vif du sujet.

Création d'une nouvelle branche

Un client nous demande une évolution du code. Afin de ne pas toucher à la branche **master** qui contient le code de référence courant, nous allons compliquer maintenant un peu les choses en créant une nouvelle branche nommée **branch-with-foo**. Cette étape a déjà été expliquée dans la partie 4 plus en détail.

```
$ git checkout -b branch-with-foo
Switched to a new branch 'branch-with-foo'
```

Nous créons immédiatement un fichier nommé **foo** dans cette branche que nous ajoutons et enregistrons dans la foulée.

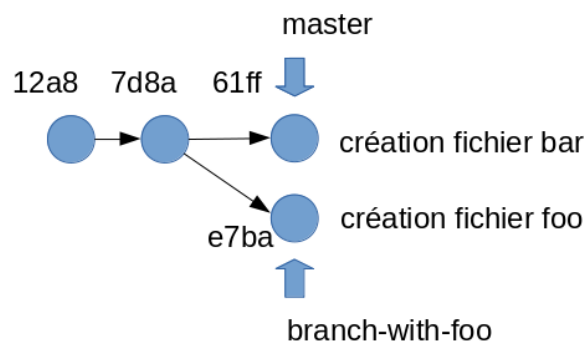
```
$ echo "this is a foo file" > foo
$ git add foo && git commit foo -m "add foo file"
[branch-with-foo d9afaa2] add foo file
1 file changed, 1 insertion(+)
create mode 100644 foo
```

Divergence des branches

Nous revenons maintenant sur **master** et nous créons un fichier **bar** que nous enregistrons aussi dans la foulée.

```
$ git checkout master
$ echo "this is a bar file" > bar
$ git add bar && git commit bar -m "add bar file"
[master 222c618] add bar file
1 file changed, 1 insertion(+)
create mode 100644 bar
```

Faisons une pause, nous venons de créer notre première divergence entre nos branches, nous avons créé un embranchement dans l'historique de nos commits, les deux derniers commits n'appartenant plus aux mêmes branches.



Le schéma présente la divergence entre la branche **master** et la branche **branch-with-foo**. La première contient un fichier **bar**, la seconde un fichier **foo**. Bien, il est temps de passer aux choses sérieuses.

Fuuuuuuuuuusion

Le besoin que nous avions de créer une nouvelle branche a disparu, le client a annulé le projet.

Bon, nous allons réintégrer les modifications de cette branche dans la branche **master**. Nous nous positionnons dans la branche **master**, ou d'une manière générale la branche à laquelle nous souhaitons réintégrer les modifications d'une autre, et nous passons la commande suivante :

```
$ git checkout master
$ git merge branch-with-foo -m "Merge branch 'branch-with-foo'"
Merge made by the 'recursive' strategy.
foo | 1 +
1 file changed, 1 insertion(+)
create mode 100644 foo
```

La sortie de la commande nous précise ce qui s'est passé : un fichier **foo** (celui de la branche **branch-with-foo**) a été créé dans la branche courante **master**.

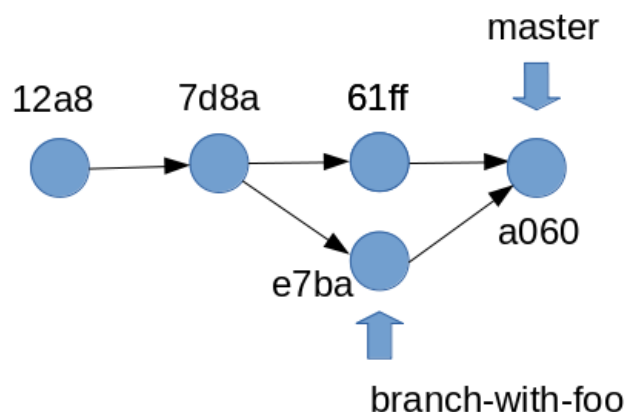


Jetons un oeil à l'historique avec la commande `git log` avec l'option `--graph` qui va nous présenter une représentation graphique de notre historique et l'option `--oneline` afin de rendre la commande moins verbeuse.

```
$ git log --graph --oneline
* 69fa060 (HEAD -> master) Merge branch 'branch-with-foo'
| \
| * d9afaa2 (branch-with-foo) add foo file
| * |222c618 add bar file
| /
* 2059805 add first subtitle
* 11cb68a (origin/master, origin/HEAD) Initial commit
```

Cette représentation est très parlante. Au niveau de l'histoire elle va de haut en bas, le haut étant le **commit** le plus récent et le plus bas le plus vieux. Une étoile (*) est un **commit**, avec son message à droite, et le nom de la branche s'il y a ambiguïté.

Nous reprenons notre dessin précédent et le faisons évoluer.



Nous avons bien fusionné la branche `branch-with-foo` dans `master`. **Fusion réussie.**

Sauver son code et ses branches

Avant de s'arrêter aujourd'hui, n'oublions pas de pousser tout ce que nous avons fait vers notre

dépôt Gitlab distant. Nous commençons par la branche **master**.

```
$ git push
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (11/11), 975 bytes | 975.00 KiB/s, done.
Total 11 (delta 2), reused 0 (delta 0)
To https://gitlab.com/chaica/merge-branches.git
11cb68a..69fa060 master -> master
```

La dernière ligne indique bien que nous avons poussé depuis notre branche **master** locale vers notre branche **master** distante présent sur notre Gitlab.

Passons à la branche **branch-with-foo**, qui, même si elle a été fusionnée dans **master**, existe toujours. Pourquoi ? Car le nom de la branche **branch-with-foo** est un pointeur, un indicateur qui désigne le dernier commit connu de cette branche. Rien de plus.

Pour changer un peu nous varions la syntaxe de notre commande **git push** en utilisant l'option **--all** afin de pousser toutes les branches locales vers le dépôt distant. Nous n'en avons qu'une ici, **branch-with-foo**.

```
$ git push --all
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for branch-with-foo, visit:
remote:   https://gitlab.com/chaica/merge-
branches/merge_requests/new?merge_request%5Bsource_branch%5D=branch-with-foo
remote:
To https://gitlab.com/chaica/merge-branches.git
[new branch]      branch-with-foo -> branch-with-foo
```

Point très intéressant, nous voyons que les lignes commençant par **remote:** proviennent du Gitlab, qui nous indiquent comment créer une demande de fusion (**Merge Request**). Inutile, nous avons déjà fusionné. Mais ce sera intéressant dans le cadre du travail collaboratif. Ce sera pour un prochain article.

Carl Chenet > merge-branches > Merge Requests > New

New Merge Request

From **branch-with-foo** into **master** [Change branches](#)

Title

Remove the **WIP:** prefix from the title to allow this **Work In Progress** merge request to be merged when it's ready.
Add [description templates](#) to help your contributors communicate effectively!

Description

Write Preview **B I " « »** **🔗** **☰** **☰** **📎** **📎** **📎**

Write a comment or drag your files here...

[Markdown](#) and [quick actions](#) are supported [📎 Attach a file](#)

Assignee [Assign to me](#)

La dernière ligne nous confirme que nous avons bien poussé la branche locale **branch-with-foo** vers la branche du dépôt Gitlab distant nommée également **branch-with-foo**.

Conclusion

Nous avons vu aujourd'hui la fusion de branches Git. Cette opération permet de récupérer le travail réalisé dans une autre branche, en divergence du code "principal" que nous conservons – comme bonne pratique dans l'industrie en général – dans la branche **master**. C'est le cas le plus courant.

Vous devriez quasi systématiquement commencer par créer une nouvelle branche quand vous envisagez d'introduire du nouveau code dans un dépôt Git, afin de ne pas travailler directement vous-même dans **master**. Pourquoi pas ? C'est ce que nous verrons dans le prochain article de cette série.

2.6. Une fusion de branches échoue

Nous allons démystifier une situation qui provoque un stress important auprès des nouveaux utilisateurs de Git : lorsqu'une fusion de branches échoue.

Il arrive malheureusement qu'une fusion de branches échoue. Ce cas se présente surtout dans l'utilisation collaborative de Git, où différentes personnes interviennent en même temps sur le code.

En effet Git est un outil très bien conçu, mais il ne prendra jamais une décision à la place de ses utilisateurs. Or, si la même partie du code est modifiée par deux utilisateurs différents, qui a raison ? Qui a tort ? Qui Git doit-il croire ?

Mise en place du conflit de fusion

Nous allons tout d'abord créer un fichier **README.md** dans un nouveau dépôt Git avec un titre et deux sections.

```
$ mkdir fusion-echoue
$ cd fusion-echoue/
$ git init .
Initialized empty Git repository in /tmp/fusion-echoue/.git/
$ vi README.md
$ cat README.md
# README

## installation

apt-get install whatever

## use

whatever param1 param3 param3
$ git add README.md && git commit README.md -m "add README.md"
add README.md
1 file changed, 9 insertions(+)
create mode 100644 README.md
```

Nous créons maintenant une branche dédiée à l'ajout d'une nouvelle section pour indiquer la licence du projet dans le **README.md**.

```
$ git checkout -b new-license
Switched to a new branch 'new-license'
$ vi README.md
$ git commit README.md -m "license section"
license section
1 file changed, 3 insertions(+)
```

Jetons un oeil à l'état actuel de notre fichier sur a branche **new-license** :

```
# README

## installation

apt-get install whatever

## use

whatever param1 param3 param3

# license
BSD
```

Un code, deux choix différents

Pendant ce temps, dans la branche **master**, un développeur semble avoir fait un choix différent au niveau de la licence.

```
git checkout master
Switched to branch 'master'
$ vi README.md
$ git commit README.md -m "license section - use gpl"
[master bf15ff3] license section - use gpl
1 file changed, 3 insertions(+)
```

Voici l'état actuel du fichier sur la branche master.

```
# README

## installation

apt-get install whatever

## use

whatever param1 param3 param3

# license
GPLv3
```

Une fusion destinée à échouer

L'événement-déclencheur du drame va être la volonté de supprimer la branche créée auparavant et de réintégrer son code dans **master**. Pour cela, rien de plus simple, nous utilisons **git merge** comme

vu dans l'article précédent.

```
$ git merge new-license
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Oups, le **CONFLICT** en majuscule nous annonce que Git a rencontré un conflit en tentant de fusionner la branche **new-license** dans la branche **master**.

La dernière phrase est très claire : résoudre les conflits est un préalable indispensable à continuer la fusion.

Résoudre le conflit... ou pas

Si vous avez déjà rencontré ce problème en utilisant Git, vous avez sûrement éprouvé ce sentiment : la grosse panique. Vous vous voyez déjà accuser des pires manipulations par vos collègues et de casser l'outil de travail commun. **DU CALME, PAS DU TOUT.**

Avant de céder à la panique, repassons la commande **git status**.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
Unmerged paths:
  (use "git add ..." to mark resolution)
both modified:   README.md
```

La sortie de la commande est très claire : un simple **git merge --abort** vous fera revenir à la situation précédant le **git merge**. Voyons cela.

```
$ git merge --abort
$ git status
On branch master
nothing to commit, working tree clean
```

Voilà, vous pouvez vous détendre et ranger votre lettre de démission. Tout va bien se passer.

Résoudre le conflit

Nous allons maintenant tenter de réellement résoudre notre conflit. Avant cela, nous devons prendre conscience de la tâche qui nous attend.

```
$ git merge new-license
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
Unmerged paths:
  (use "git add ..." to mark resolution)
both modified:   README.md
```

La sortie de la commande `git status` nous indique qu'un seul fichier pose problème par la mention sur la dernière ligne `both modified: README.md`.

Ce n'est pas nos quelques lignes dans le `README.md` qui vont nous décourager. Jetons donc un oeil au fichier en question.

```
$ cat README.md
# README

## installation

apt-get install whatever

## use

whatever param1 param2 param3

## license
<<<<<< HEAD
GPLv3
=====
BSD
>>>>>> new-license
```

Surprise, Git nous a maché le travail.

Dans la section license du fichier, nous voyons que la partie entre `<<<<<< HEAD` et `=====` provient de `HEAD`, qui pointe actuellement sur `master`, donc le code provenant de `master`. Vous avez oublié ce qu'est `HEAD` ? On en parle dans la partie 4.

À l'opposé, la partie entre `=====` et `>>>>>> new-license` provient donc de la branche `new-license`. C'est maintenant à vous de jouer et de faire un choix, Git ne le fera pas à votre place, une décision humaine est attendue.

Après consultation avec vos collègues, vous choisissez la licence GPLv3. Vous allez donc éditer le code de la section license pour ne laisser que la partie suivante :

```
## license
GPLv3
```

La commande `git status` précédente nous indiquait la marche à suivre une fois le fichier édité.

```
Unmerged paths:
  (use "git add ..." to mark resolution)
  both modified:   README.md
```

Nous suivons donc ces instructions à la lettre.

```
$ git add README.md
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)
```

Le conflit est résolu. Nous allons maintenant utiliser `git commit` pour terminer la fusion.

```
$ git commit -a -m "use GPLv3 license. merge from new-license branch"
```

Si vous n'utilisez que la commande `git commit` sans argument, votre éditeur de texte s'ouvre et propose d'enregistrer le message de commit suivant :

```
Merge branch 'new-license'
```

Conclusion

Nous avons brillamment résolu notre premier conflit de fusion entre deux branches. Il était franchement facile à régler, mais il nous a permis de détailler toutes les étapes nécessaires. Bien évidemment nous pouvons avoir des conflits bien plus sévères à régler, mais nous avons vu comment interrompre une fusion, ce qui nous aidera quoiqu'il arrive à revenir à un état stable du dépôt.

S'engager dans une fusion complexe réclame une très bonne connaissance du code manipulé et de ce qu'on fait vos collègues, pour ne pas écraser du code utile par inadvertance.

Chapitre 3. 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.

3.1. 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.

3.2. 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.

3.3. 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
```

3.4. 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
```

3.5. Les modules Puppet

Les modules complémentaires à Puppet peuvent être téléchargés sur le site [puppetlabs](http://puppetlabs.com).

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.

3.6. 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>

3.7. 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
```

3.8. 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","ntpddate","snmpd","snmptrapd"]:  
    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/default",  
}
```



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.

Chapitre 4. 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.

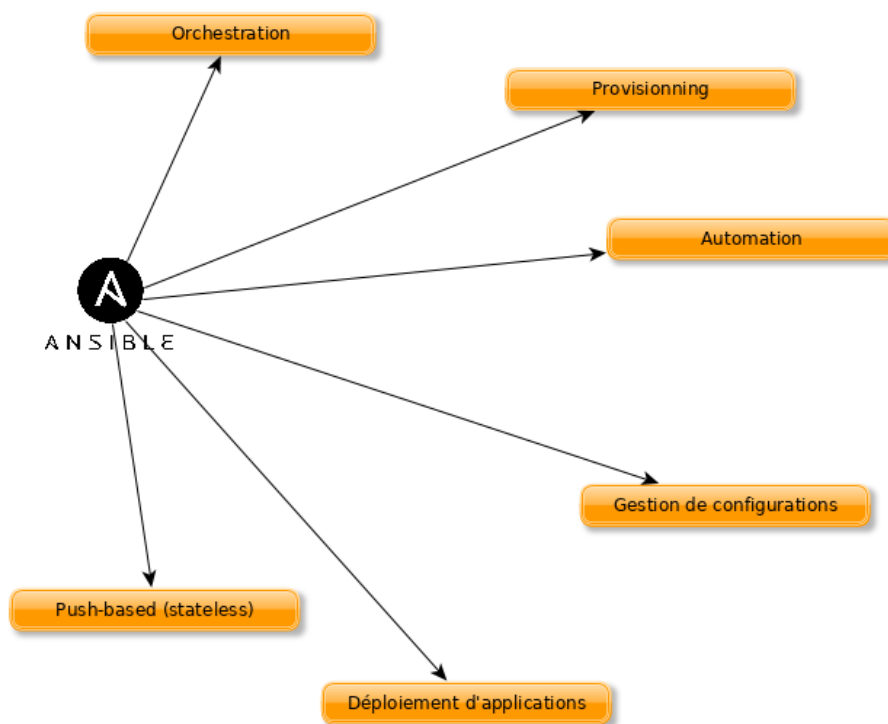


Figure 2. Les fonctionnalités d'Ansible



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

4.1. 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 exécuter (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** : ils sont utilisés pour provoquer un arrêt ou un redémarrage d'un service en cas de changement.

4.2. 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 'webserver' group

## [webserver]
## 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
```

4.3. 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
<code>-a 'arguments'</code>	Les arguments à passer au module.
<code>-b -K</code>	Demande un mot de passe et lance la commande avec des privilèges supérieurs.
<code>--user=utilisateur</code>	Utilise cet utilisateur pour se connecter à l'hôte cible au lieu d'utiliser l'utilisateur courant.
<code>--become</code> <code>-user=utilisateur</code>	Exécute l'opération en tant que cet utilisateur (défaut : <code>root</code>).
<code>-C</code>	Simulation. Ne fait pas de changement sur la cible mais la teste pour voir ce qui devrait être changé.
<code>-m module</code>	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...

4.4. 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]---+
|
|      .  .
|     = . +
|    + 0 *   . +.0
|   o * S. . *o*.
|   o * .o ..+=
|      o. .000E
|      .+ 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 clé privée est créée et téléchargée sur le poste local.

Ajout de la clé 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
```

4.5. 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 lieu 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"
}
```

4.6. 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 miroir1.local.lan miroir1"
        state: present
    - name: configurer le noeud 2 dans /etc/hosts
      lineinfile:
        dest: /etc/hosts
        line: "52.14.125.109 miroir2.local.lan miroir2"
        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

...
```

4.7. 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
```

4.8. 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.

Chapitre 5. 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.

5.1. 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.

```
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.

5.2. 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 }}"
```

5.3. Les conditions



Plus d'informations
sur [playbooks_conditionals.html](http://docs.ansible.com/ansible/latest/playbooks_conditionals.html)

sur <http://docs.ansible.com/ansible/latest/>

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 == ""
```

5.4. 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.

5.5. 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 déclencher 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"
```

5.6. 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.

5.7. 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
```

5.8. 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
```

Chapitre 6. Ansible 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.

6.1. 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 peu de nettoyage (suppression des anciennes versions)

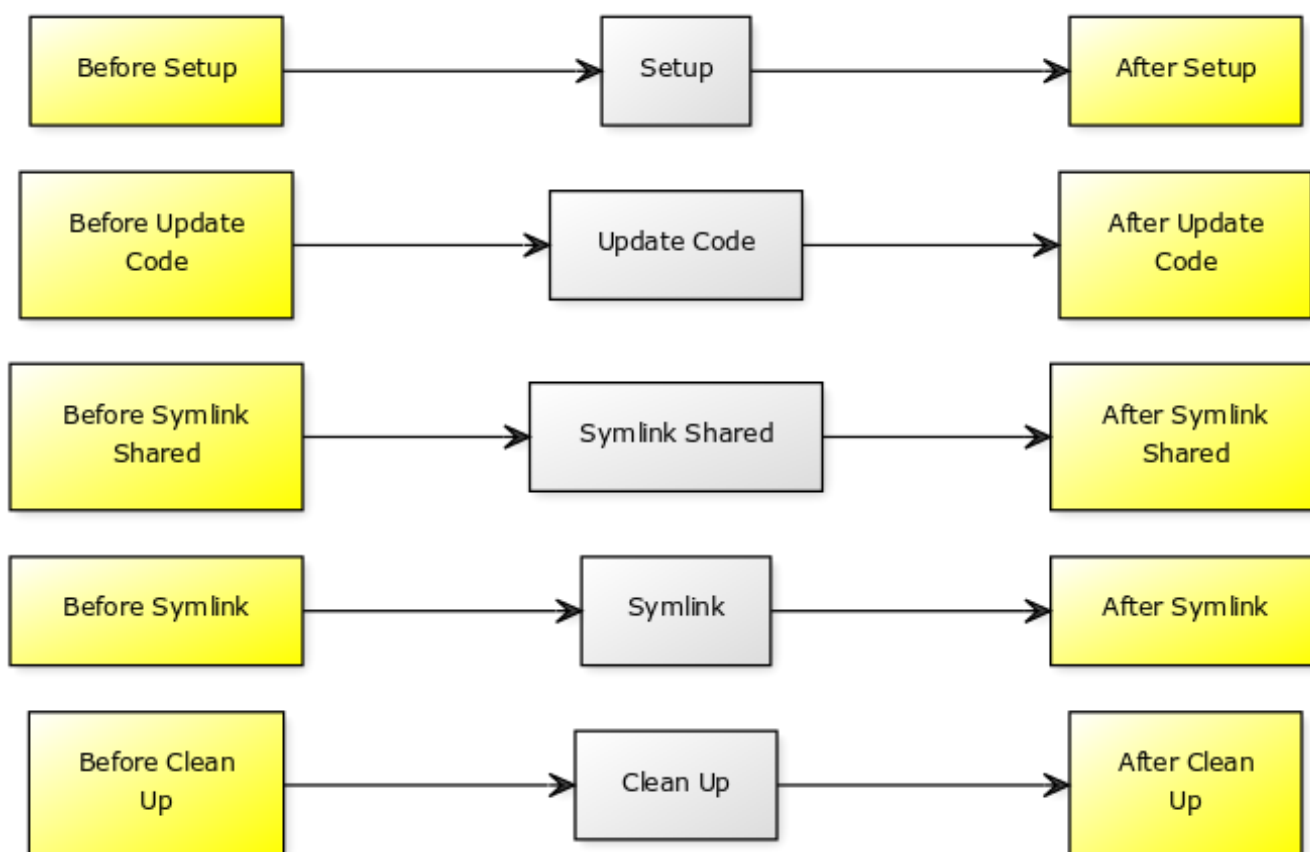


Figure 3. Etapes d'un déploiement

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](#).

6.2. 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

6.3. 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**.


```
---
- 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 facilite grandement la tâche en s'occupant de l'installation et la configuration d'Apache.

6.4. Module 3 : Déployer le logiciel

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

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 }}"

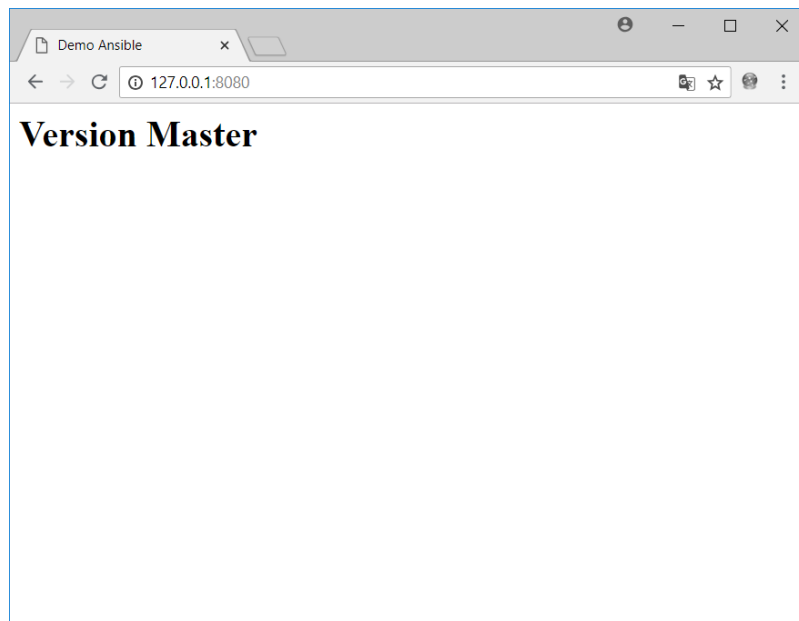
  roles:
    - { role: ansistrano.deploy }
```

```
[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

6.5. 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


```
---
- 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**

6.6. 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.

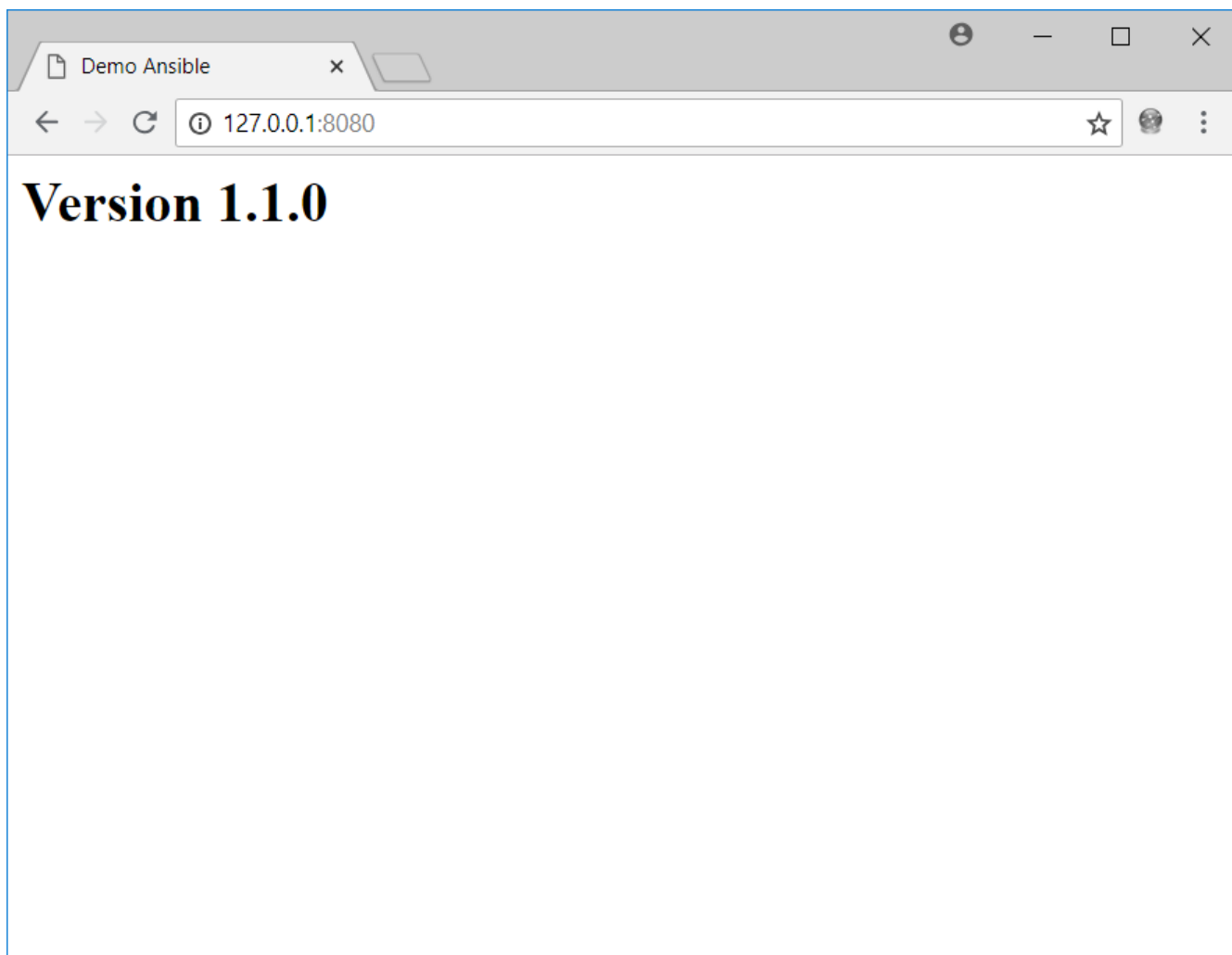
Exercice 5.1 : Déployer une branche

- 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** :

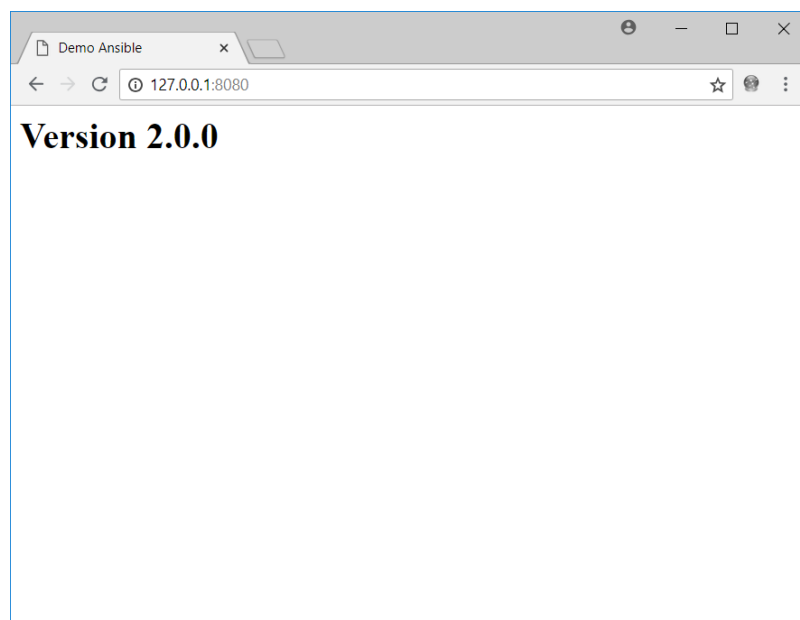
```

---
- 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.



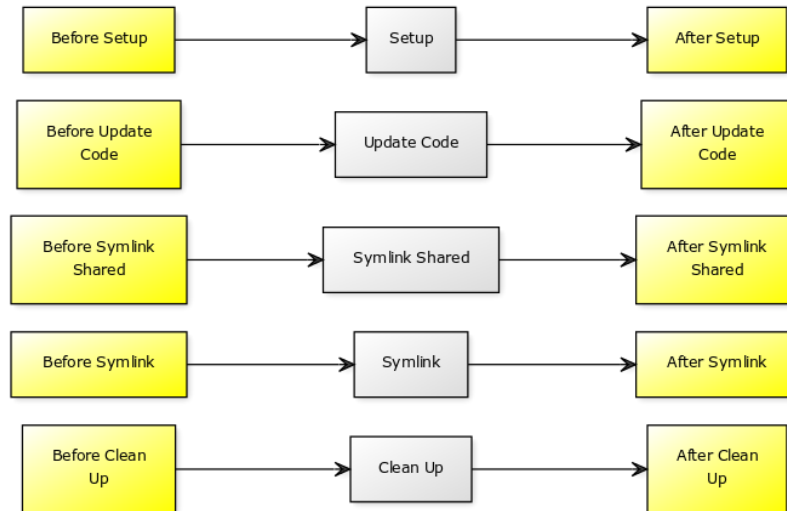
6.7. Module 6 : Actions entre les étapes de déploiement

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]
```


Chapitre 7. 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.

7.1. Installation



Rundeck utilisant le protocole 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
```

7.2. 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.

7.3. 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.

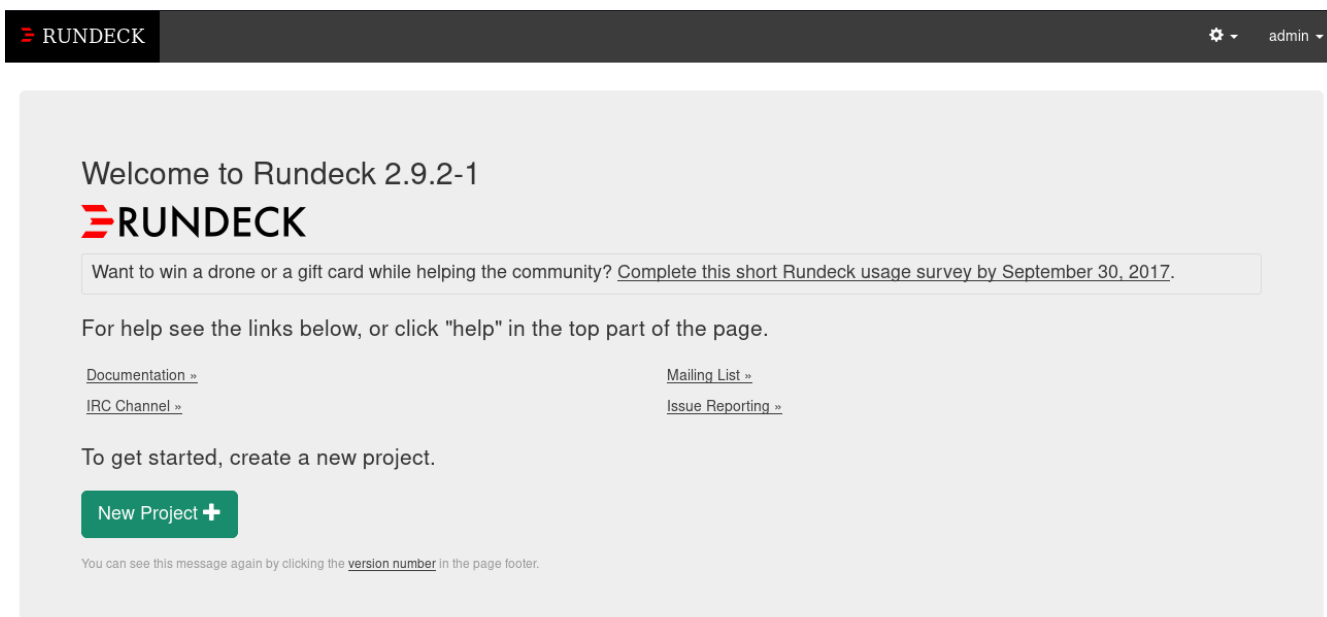


Figure 4. La page d'accueil de RunDeck

Créer un projet

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

The image shows the 'Create a new Project' form in the Rundeck interface. The form is titled 'Create a new Project' and has a dark header. It contains several sections: 'Project Name' with a text input field, 'Description' with a text input field, and 'Resource Model Source'. The 'Resource Model Source' section includes a list of sources, with the first one being 'File'. It shows details for the 'File' source: 'Format: resourcexml', 'File Path: /var/rundeck/projects/\${project.name}/etc/resources.xml', 'Generate: Yes', and 'Include Server Node: Yes'. There are 'Delete' and 'Edit' buttons for each source. Below the sources is an 'Add Source +' button. The 'Execution Mode' section is at the bottom, with a checkbox for 'Disable Execution'.

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.

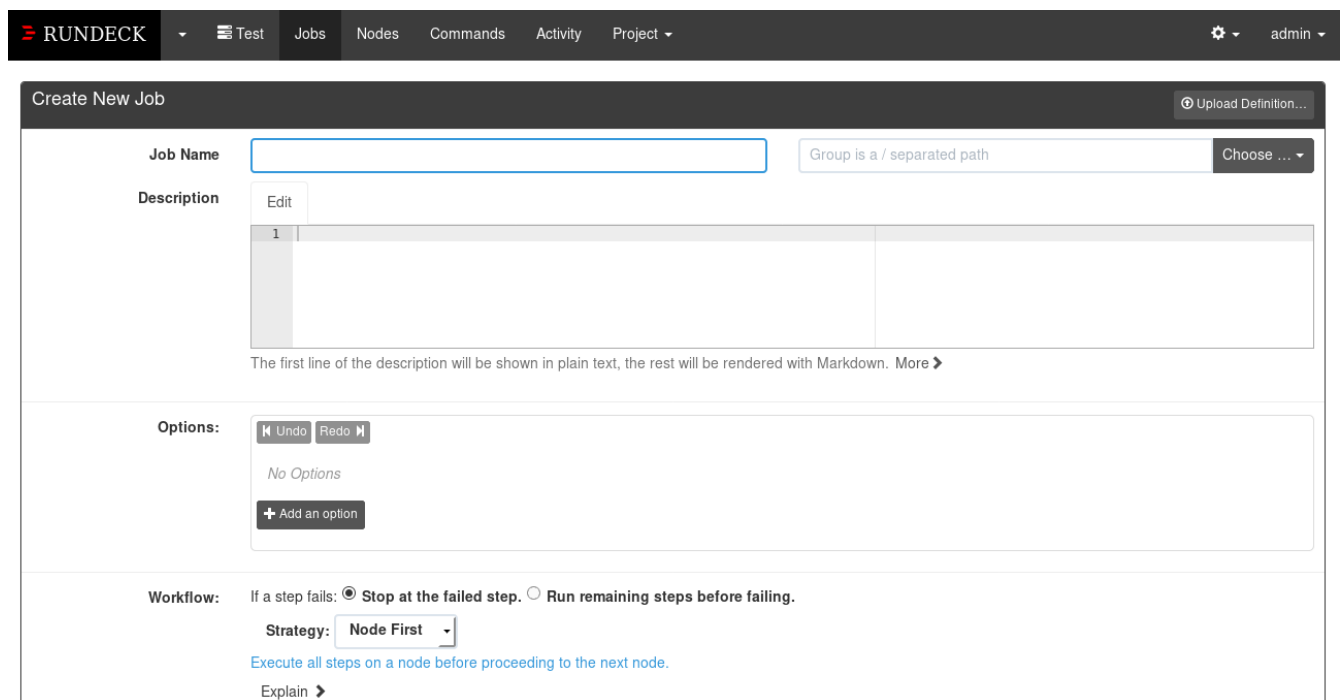
The screenshot shows the 'Create New Job' interface in RunDeck. At the top is a dark navigation bar with the RunDeck logo and menu items: Test, Jobs, Nodes, Commands, Activity, and Project. On the right of the bar are a settings icon and the user 'admin'. Below the navigation bar is the 'Create New Job' form. It has a header bar with 'Create New Job' on the left and an 'Upload Definition...' button on the right. The form is divided into several sections. The first section is for 'Job Name' with a text input field and a 'Group is a / separated path' dropdown menu with a 'Choose ...' button. Below this is the 'Description' section, which includes an 'Edit' button and a table with one row labeled '1'. A note below the table states: 'The first line of the description will be shown in plain text, the rest will be rendered with Markdown. More >'. The next section is 'Options:', which contains 'Undo' and 'Redo' buttons, the text 'No Options', and an 'Add an option' button. The final section is 'Workflow:', which includes radio buttons for 'Stop at the failed step.' (selected) and 'Run remaining steps before failing.', a 'Strategy:' dropdown menu set to 'Node First', and a link 'Execute all steps on a node before proceeding to the next node.' with an 'Explain >' link below it.

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**.

Add New Option

Option Type
Text

Option Name
Option Name

Description

1

The description will be rendered with Markdown.

Default Value
Default value

Input Type

☒ Plain text
☐ Date The date will pass to your job as a string formatted this way: mm/dd/yy HH:MM
☐ Secure Password input, value exposed in scripts and commands.
☐ Secure Remote Authentication Password input, value not exposed in scripts or commands, used only by Node Executors for authentication.

+ Secure input values are not stored by Rundeck after use. If the exposed value is used in a script or command then the output log may contain the value.

Allowed Values

☒ List
☐ Remote URL

Comma separated list

Restrictions

☒ None Any values can be used
☐ Enforced from Allowed Values

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.

Chapitre 8. 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.

8.1. 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 dépôt 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'adresse :

- <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'adresse :

- <http://IPSERVEUR:8080/jenkins>

8.2. 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
```

8.3. 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 :

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

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

Please copy the password from either location and paste it below.

Administrator password

Continue

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`.

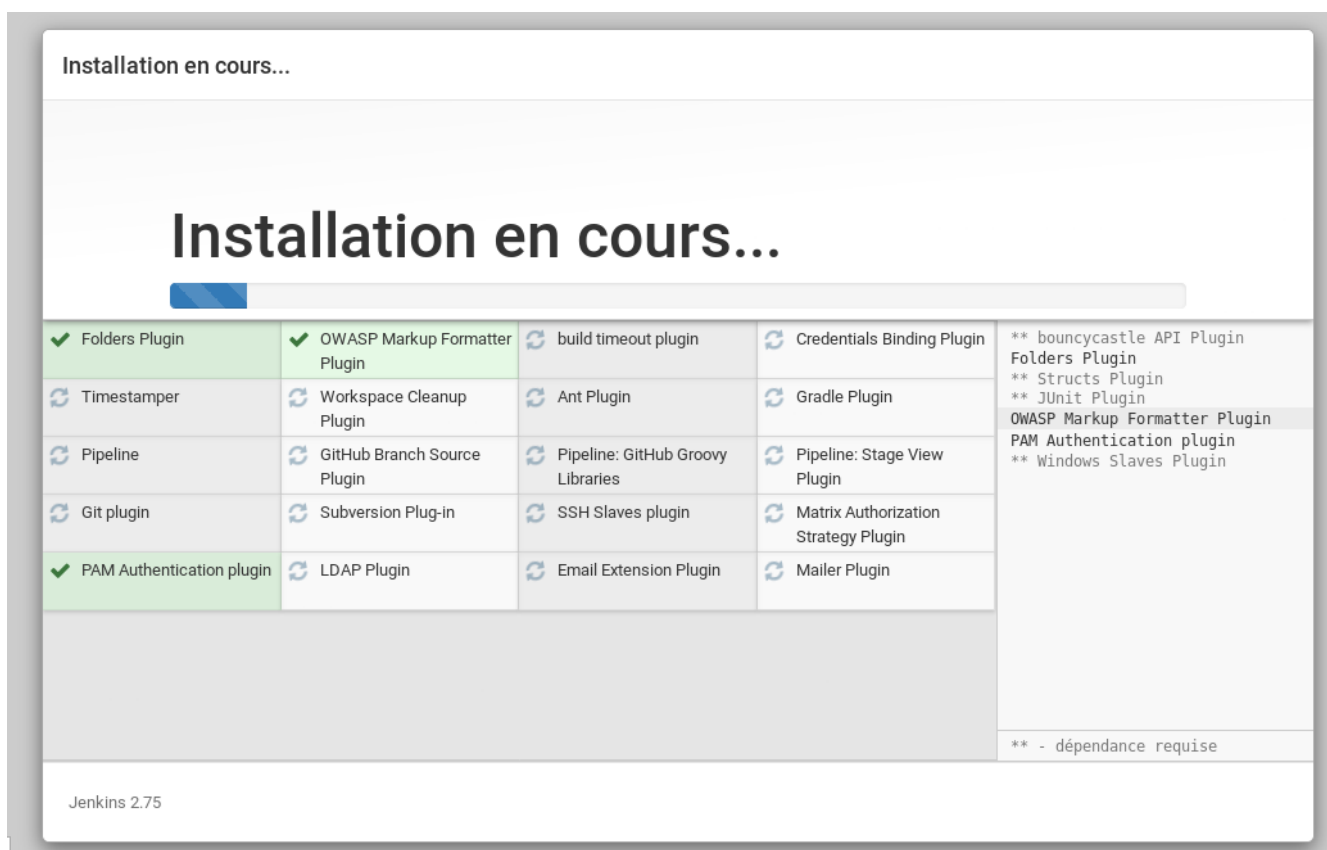
```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

Utiliser le mot de passe pour se 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 :

Démarrage

Create First Admin User

Username:

admin

Password:

••••••••

Confirm password:

••••••••

Full name:

Admin Formatux|


E-mail address:

contact@formatux.fr

Jenkins 2.75

[Continuer en tant qu'Administrateur](#) [Sauver et Terminer](#)

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

 **Jenkins**

search

Admin Formatux | log out

Jenkins

[ENABLE AUTO REFRESH](#)

New Item

People

Build History

Manage Jenkins

My Views

Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Welcome to Jenkins!

Please [create new jobs](#) to get started.

[add description](#)

Page generated: Aug 23, 2017 6:08:41 PM UTC [REST API](#) [Jenkins ver. 2.75](#)

8.4. 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.

Jenkins » Configure Global Security

Access Control

Security Realm

- ☐ Delegate to servlet container
- ☒ Jenkins' own user database
- ☐ Allow users to sign up
- ☐ LDAP
- ☐ Unix user/group database

Authorization

- ☐ Anyone can do anything
- ☐ Legacy mode
- ☐ Logged-in users can do anything
- ☒ Matrix-based security

	Overall	Credentials	Agent	Job	Run	View	SCM
User/group		Manage Domains					
Administrator	Read	Create	Delete	Update	View	Build	Configure
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Save Apply

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**.

8.5. Ajouter une tâche d'automatisation simple

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

Jenkins

search Admin Formatux | log out

Enter an item name

> This field cannot be empty, please enter a valid name

- Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- External Job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.
- Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- GitHub Organization**
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

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**, sélectionner l'option **Execute shell**.

Saisir la commande suivante dans le champ texte :

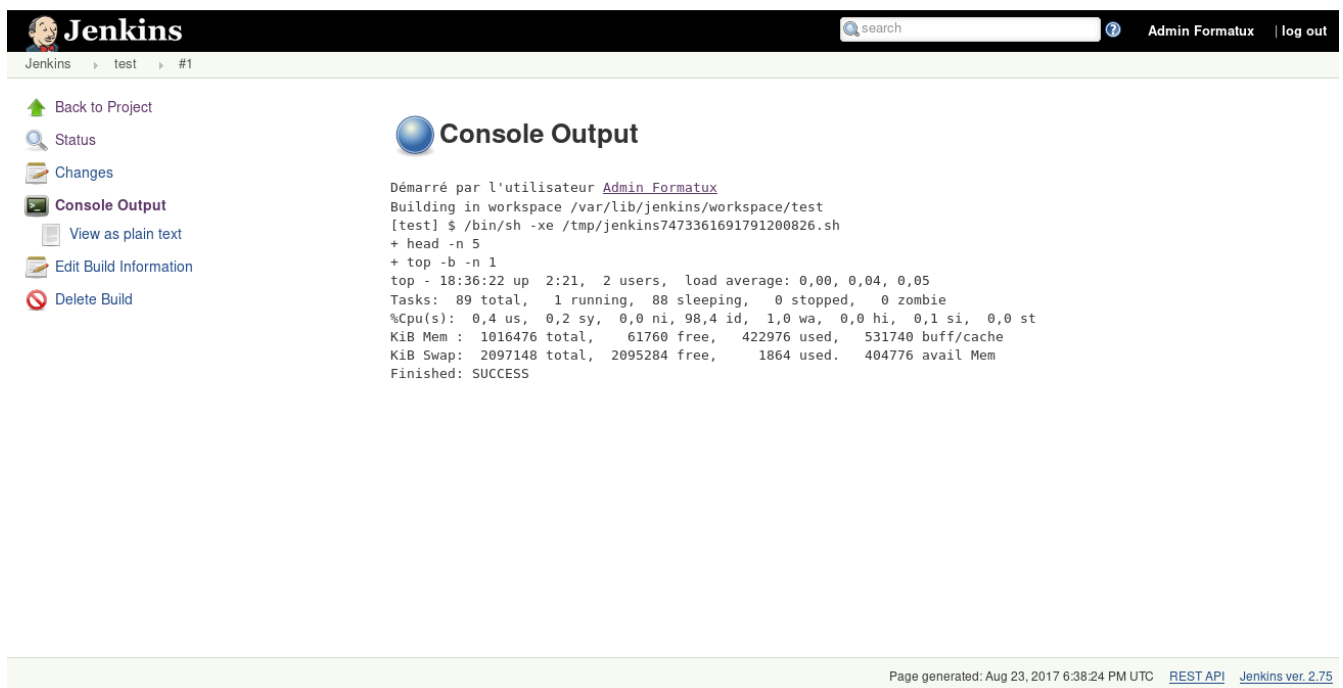
```
top -b -n 1 | head -n 5
```

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 exécutée, vous verrez l'historique du Build. Cliquer sur la première tâche

pour visualiser les résultats.



The screenshot displays the Jenkins web interface. At the top, there's a navigation bar with the Jenkins logo, a search bar, and links for 'Admin Formatux' and 'log out'. Below the navigation bar, the breadcrumb trail shows 'Jenkins > test > #1'. On the left sidebar, there are links for 'Back to Project', 'Status', 'Changes', 'Console Output' (which is highlighted), 'View as plain text', 'Edit Build Information', and 'Delete Build'. The main content area is titled 'Console Output' and shows the following text:

```
Démarré par l'utilisateur Admin.Formatux
Building in workspace /var/lib/jenkins/workspace/test
[test] $ /bin/sh -xe /tmp/jenkins7473361691791200826.sh
+ head -n 5
+ top -b -n 1
top - 18:36:22 up 2:21, 2 users, load average: 0,00, 0,04, 0,05
Tasks: 89 total, 1 running, 88 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,4 us, 0,2 sy, 0,0 ni, 98,4 id, 1,0 wa, 0,0 hi, 0,1 si, 0,0 st
KiB Mem : 1016476 total, 61760 free, 422976 used, 531740 buff/cache
KiB Swap: 2097148 total, 2095284 free, 1864 used, 404776 avail Mem
Finished: SUCCESS
```

At the bottom of the page, there is a footer that says 'Page generated: Aug 23, 2017 6:38:24 PM UTC' followed by links for 'REST API' and 'Jenkins ver. 2.75'.

8.6. Sources

Source principale : [howtoforge](#).

Chapitre 9. DocAsCode : le format AsciiDoc

9.1. 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).

9.2. 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`

9.3. Références

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

Chapitre 10. Infrastructure as Code : Terraform

10.1. 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** (**HashiCorp Configuration Language**),
- **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 ressources :

```
# 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>.

10.2. La HCL

Le langage HashiCorp Configuration Language 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**.

10.3. Les providers

Terraform est utilisé pour gérer des ressources qui peuvent être des serveurs physiques, des VM, des équipements réseaux, des conteneurs, 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 !

10.4. 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 ressourcecetype.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 régression 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`.

10.5. 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.

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

10.7. 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 deployment 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>

10.8. 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 [Ikkoula](#) 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

Index

@

--set-upstream, [16](#)

~1, [25](#)

A

augeas, [47](#)

automation, [7](#)

B

branch, [14](#)

build, [6](#)

C

CI, [7](#)

Continuous Integration, [7](#)

change, [6](#)

commit, [14](#)

D

DSL, [8](#)

detached head, [26](#)

devops, [6](#)

dépôt, [10](#)

G

git, [10](#)

git add, [14](#)

git add --patch, [19](#)

git branch, [27](#)

git checkout, [25](#)

git checkout -b, [27](#)

git clone, [12](#)

git commit, [15](#)

git config, [13](#)

git log, [16](#)

git log --graph --oneline, [32](#)

git merge, [31](#)

git merge --abort, [37](#)

git push, [16](#)

git push --all, [33](#)

git remote, [16](#)

git status, [14](#)

H

HCL, [113](#)

HEAD, [25](#)

hunk, [19](#)

I

idempotence, [7](#)

intégration continue, [7](#)

J

Jenkins, [99](#)

M

Merge Request, [33](#)

O

orchestration, [7](#)

P

provisionning, [7](#)

puppet, [40](#)

R

Rundeck, [93](#)

run, [6](#)

S

stage, [15](#)