UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
iut Créteil Vitry

iut réseaux & télécoms Créteil-Vitry

TP N°1

SUJET :

INFRASTRUCTURE AS CODE

UE 3.6
INFRASTRUCTURE SYSTEMES ET RESEAUX

BRICE AUGUSTIN

Durée : 6 heures

Version : 1.2

Création : 05/2018
Révisions : 06/2018, 01/2019, 03/2019

Sommaire

1. Introduction 3
1.1. Contexte 3
1.2. Maquette 4
1.3. Travail demandé 5
2. Préparation 7
2.1. Hôte 7
2.2. Serveur Git 8
3. Construction de la VM : aspects généraux 10
3.1. Packer 10
3.2. Configuration réseau 12
3.3. Configuration système 13
4. Construction de la VM : aspects spécifiques 15
4.1. Firewall 15
4.2. Load balancer 15
4.3. Serveurs Web 15
4.4. Serveur SQL 16
4.5. Collecteur 16
5. Recette 18
6. Mémos 20
6.1. Git 20
6.2. Packer 23
6.3. VirtualBox 26
6.4. Divers 28

1. Introduction

1.1. Contexte

Vous êtes administrateur système et réseau pour une entreprise qui s'apprête à commercialiser une application de *microblogging* anonyme. Cette application est accessible via une interface Web très simple qui permet aux utilisateurs de visualiser les derniers messages publiés, et d'en poster de nouveaux.

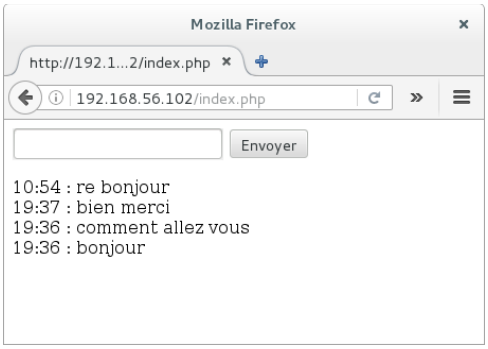

Fig. 1 *Interface Web de l'application de microblogging*

En interne, ces messages sont stockés dans une base de données (*back-end*), qui n'est pas accessible directement par les utilisateurs.

Un *load balancer* permet de répartir les requêtes sur les deux serveurs Web (*front-end*).

Enfin, un pare-feu protège l'accès à l'application en filtrant le trafic émis et reçu.

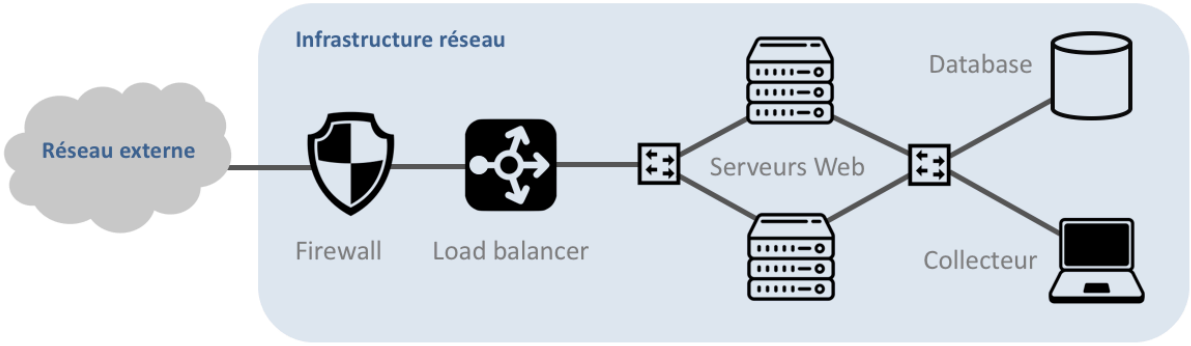

Fig. 2 *Infrastructure réseau de l'application*

L'écriture du code de l'application (en `PHP`, `SQL` et `Bash`) a été confiée à une équipe de développeurs qui vient de terminer une première version. Il faut maintenant passer à la phase de tests.

Dans un premier temps, vous êtes chargés de préparer une réplique de l'environnement dans lequel l'application sera déployée. Vous allez donc construire une infrastructure réseau composée de ces cinq éléments :

- Un firewall
- Un *load balancer*
- Deux serveurs Web
- Un serveur de base de données
- Un collecteur de *logs*

1.2. Maquette

On souhaite automatiser le déploiement de cette infrastructure virtuelle en utilisant une approche "Infrastructure as code" : pour construire chaque VM, on a recours à des fichiers contenant toutes les instructions nécessaires, plutôt qu'à un traitement manuel.

Le déploiement s'appuie sur les technologies suivantes :

- Un logiciel de gestion de version (*version control system*) qui conserve tous les fichiers de configuration. Vous utiliserez l'outil le plus populaire actuellement, à savoir `Git`

- Un logiciel de préparation de VM, qui construit et provisionne l'infrastructure virtuelle en respectant les instructions données dans des *template* de configuration. Dans ce TP, vous utiliserez `Packer`
- Un hyperviseur qui gère les VM et leur interconnexion via des réseaux virtuels

1.3. Travail demandé

Le projet est un travail collaboratif qui se déroule en deux phases.

Dans la première phase, illustrée par la Fig. 3, chaque équipe choisit *un des cinq* éléments de l'infrastructure de la Fig. 2 : firewall, *load balancer*, serveurs Web, *database* ou collecteur.

Au terme de cette phase, l'équipe devra déposer :

- Un fichier de configuration `Packer` permettant de construire leur VM
- Tous les fichiers de configuration nécessaires au fonctionnement de leur élément

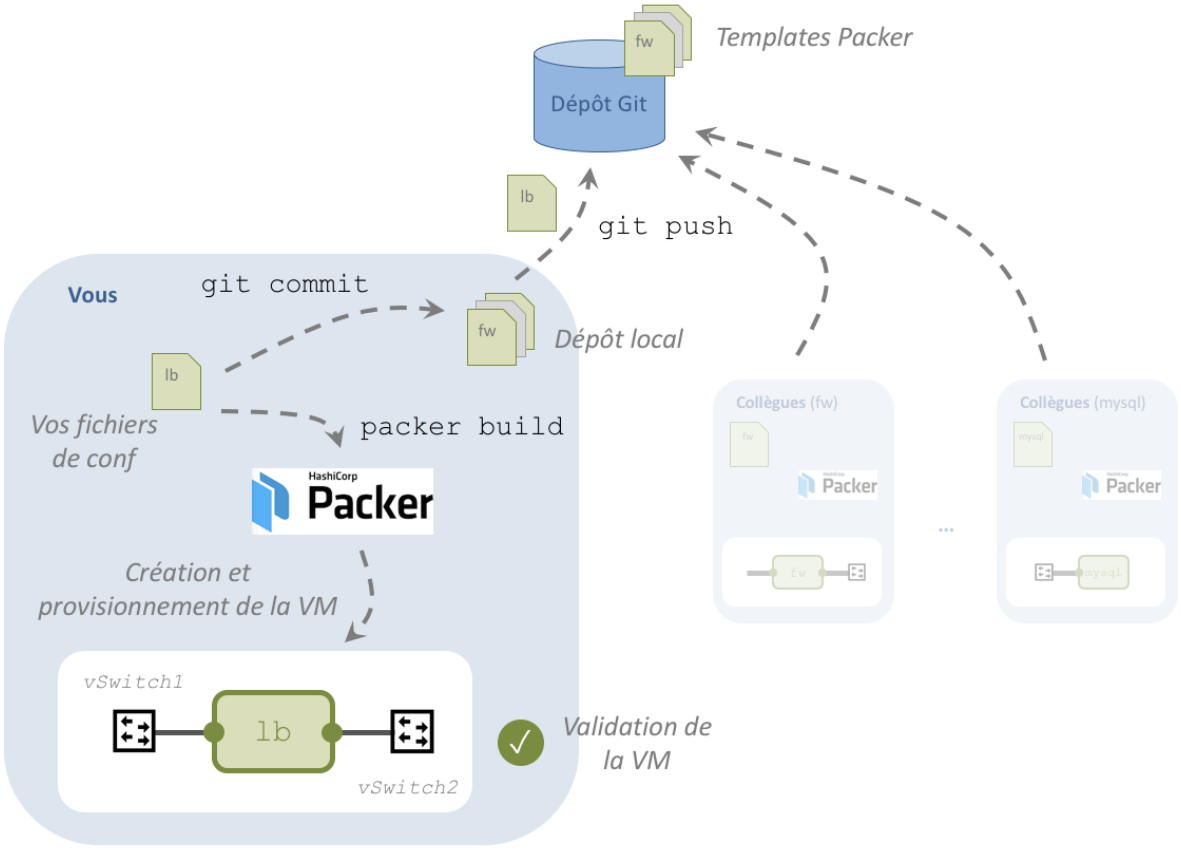

Fig. 3 *Développement de notre infrastructure programmable (Phase 1)*

Dans la seconde phase, illustrée par la Fig. 4, chaque binôme récupère la totalité du dépôt `Git` (donc, sa contribution et celle de toutes les autres équipes) et construit sa propre instance de l'infrastructure virtuelle, de manière entièrement automatisée.

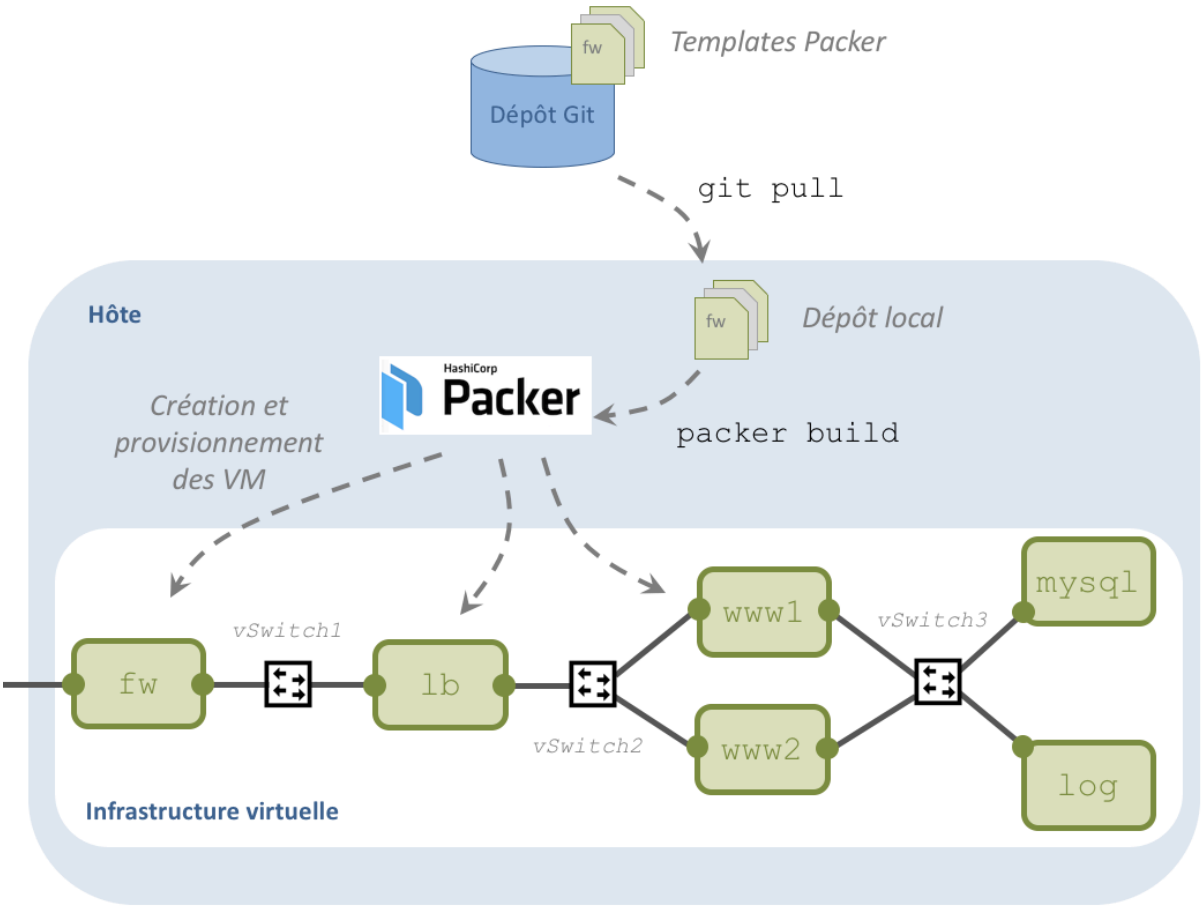

Fig. 4 *Infrastructure programmable (infrastructure as code), Phase 2*

2. Préparation

2.1. Hôte

Vous travaillerez obligatoirement sur un des PC d'extrémité, sous `Debian Linux` et avec l'hyperviseur `VirtualBox`.

Vous ne devez en aucun cas utiliser le PC du milieu pour préparer et tester votre configuration[1].

Téléchargez l'image `debian-stretch.ova`[2] depuis le serveur FTP de l'IUT et placez-la dans le dossier personnel de `etudiant`.

Cette image servira de socle commun pour construire toutes les VM du projet.

Vous ne devez pas l'importer maintenant ! C'est `Packer` qui s'en occupera.

2.2. Serveur Git

Pour mettre votre travail en commun, vous utiliserez le serveur `Git` hébergé par votre prof. Pour l'utiliser, installez simplement le paquetage `git` sur votre PC et demandez l'adresse IP du serveur à votre prof.

Pour rappel, vous devez travailler uniquement en tant qu'utilisateur `etudiant`. *Travailler en `root` n'est pas nécessaire … et ne vous apportera que des ennuis.*

Téléchargez le fichier `id_rsa`[3] sur le serveur Web de votre prof. Copiez ce fichier dans votre dossier `.ssh` et vérifiez que vous pouvez maintenant vous connecter en SSH sans mot de passe.

Clonez le dépôt* `app.git` puis déplacez-vous dans le dossier créé. Observez son organisation. Il contient :

- Un dossier par élément de l'infrastructure (`fw`, `lb`, `www1`, etc.).

[1] Cela garantit que tous les binômes travaillent sur des PC strictement identiques.

[2] Il s'agit d'une installation minimaliste de Debian et compatible avec `Packer`*.

[3] Rappel : il s'agit d'une clé privée RSA …

Pour le moment, chaque dossier ne contient que le code source spécifique à chaque élément (Tableau 1). Vous n'avez pas besoin de comprendre ce code pour réaliser la maquette, mais rien ne vous empêche d'y jeter un œil …

- Un dossier `exemple`.

  Vous allez vous en inspirer pour construire votre propre fichier configuration `Packer`.

- Un script de construction de l'application (`build.sh`).

  Vous ferez appel à ce script dans la dernière phase. Pour le moment, il ne vous est d'aucune utilité.

- Un script de nettoyage (`clean.sh`).

  Vous comprendrez son utilité dans la partie suivante.

Vous devez travailler uniquement sur le dossier de l'élément dont vous avez la charge (par exemple dans `fw/`). *Vous ne devez pas toucher aux autres dossiers (c'est le travail de vos collègues) !*

VM	Fichier	Rôle
`fw`	`fw.sh`	*Active le filtrage de trafic avec iptables*
`lb`	`haproxy.cfg`	*Configure haproxy pour répartir la charge sur les deux serveurs Web*
`www1` `www2`	`index.php`	*Code PHP du site Web*
`mysql`	`app.sql`	*Crée et initialise la base de données*
`log`	`monitor.sh`	*Supervise les serveurs Web*

Tableau 1 *Code source de l'application*

Pour distinguer facilement vos contributions de celles de vos collègues, personnalisez la configuration* de Git en indiquant votre nom et adresse email.

Commencez par créer un fichier `{fw,lb,www1,…}.json` (*vide*) dans votre dossier, puis ajoutez-le* au dépôt Git. Indiquez, comme message de *commit* : "`création du fichier de conf Packer`".

Attendez que d'autres collègues parviennent à cette étape, puis faites un `pull`* sur le dépôt. Vous verrez alors apparaitre les fichiers `.json` de vos collègues dans votre dépôt local !

Affichez les contributions du dépôt* pour vous rendre compte de qui a fait quoi depuis le démarrage du projet.

3. Construction de la VM : aspects généraux

Cette partie vous fournit toutes les indications dont vous aurez besoin pour réaliser votre fichier de configuration `Packer`.

Quand votre travail sera terminé, ce fichier contiendra toutes les instructions permettant une création entièrement automatisée de votre VM.

Lisez cette partie en totalité (jusqu'à la Synthèse 1) avant de vous lancer !

3.1. Packer

Prenez quelques minutes pour étudier le fichier de configuration `exemple/dns.json`, comprendre sa syntaxe*, ainsi que le rôle d'un builder* et des provisionners*.

Il s'agit de la clé de voûte de `Packer`. Vous devez maitriser cet aspect pour réaliser ce projet !

Votre travail consiste donc à écrire un fichier de configuration `Packer` contenant toutes les instructions pour construire une VM qui respecte le cahier des charges.

Nous vous conseillons de réaliser votre fichier en plusieurs étapes. En particulier, respectez le cycle de développement suivant :

1. Écriture d'une partie du fichier de configuration :

   Par exemple, commencez avec une configuration contenant uniquement un builder, mais pas de provisionner (indiquez `"provisioners": []`).

2. Test en local :

   Vérifiez que vous pouvez construire une VM avec ce fichier. Corrigez les erreurs si ce n'est pas le cas. Si `Packer` parvient à construire la VM (pas de message d'erreur en rouge …), démarrez-la avec la GUI de `VirtualBox` et vérifiez que sa configuration est comme vous l'espérez.*

3. `Commit` et `push*` sur le serveur Git :

   Cette partie est importante pour garder une trace des modifications apportées à la configuration, et pour les partager avec vos collègues. N'oubliez pas d'indiquer un message de 'commit' précis, qui résume vos contributions.

4. Recommencez !

   Cette fois en ajoutant les provisionners nécessaires pour répondre au cahier des charges (adressage, paquetages, etc.).

   Attention, avant de passer au cycle suivant, vous devez faire un peu de nettoyage[4] :

```
# Supprimer la VM de VirtualBox
# et effacer tous ses fichiers
VBoxManage unregistervm fw --delete

# Supprimer le dossier créé par Packer
rm -rf fw/output-virtualbox-ovf/
```

[4] Voir le script `clean.sh`

3.2. Configuration réseau

Le premier aspect à configurer est le réseau. Chaque VM doit s'insérer dans le plan d'adressage de la Fig. 5.

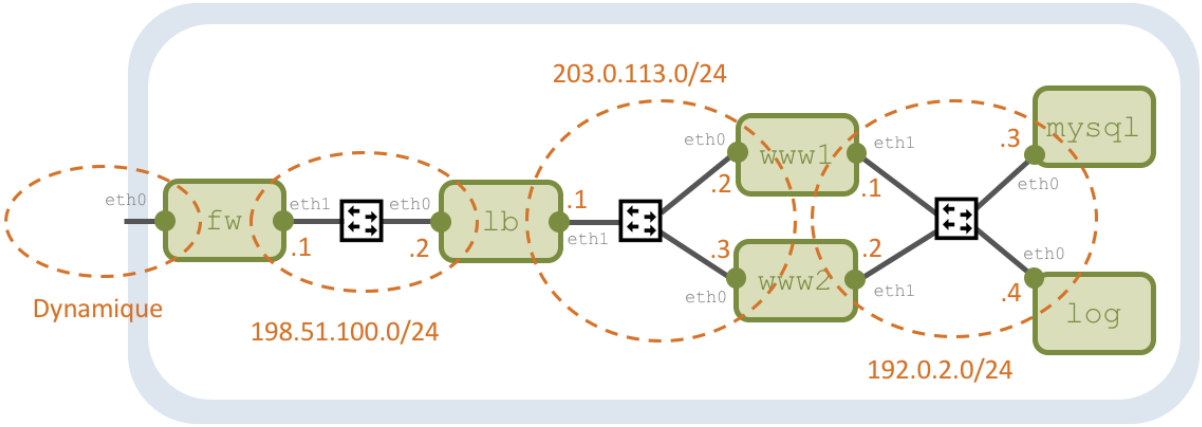

Fig. 5 *Plan d'adressage*

Toutes les cartes doivent être configurées en mode `Réseau interne` et branchées sur le switch virtuel (*vSwitch*) adéquat. Les noms des *vSwitch* sont indiqués dans la Fig. 4.

Seule exception : la première carte réseau de `fw` qui doit être configurée en mode `Bridge` pour permettre un accès à l'application depuis l'extérieur.

Les cartes réseau doivent être configurées dans le *builder* de votre fichier de configuration `Packer`

Les cartes doivent être configurées en adressage persistant*.

Soyez rigoureux ! Si vous n'utilisez pas les noms et adresses indiqués, les VM ne pourront pas communiquer ensemble lors de la phase d'intégration (phase 2).

Conseils :

Dans le dossier de votre dépôt local, créez un fichier `interfaces` avec la configuration IP adéquate. Utilisez un *provisionner** de type `file` pour envoyer ce fichier dans le répertoire `/etc/network` de la VM.

N'oubliez surtout pas d'ajouter ce fichier au dépôt Git* pour que vos collègues puissent en bénéficier !

3.3. Configuration système

Votre VM doit porter le nom indiqué dans la Fig. 4. Votre configuration `Packer` doit donc renommer* la VM pendant sa construction.

Conseils :

Là encore, vous pouvez vous appuyer sur des *provisionners* de type `file` pour envoyer des fichiers déjà prêts, ou sur des *provisionners* de type `shell` pour écraser ou remplacer le contenu des fichiers d'origine. Vous êtes libres de choisir, mais n'oubliez pas que le renommage nécessite deux étapes …

Enfin, chaque VM requiert une liste de paquetages[5] spécifiques :

VM	Paquetages	Rôle

[5] Le proxy de l'IUT est déjà configuré sur l'image OVA fournie. Ouf.

`fw`	`tcpdump`	*Analyseur réseau en CLI*
`lb`	`haproxy`	*Load balancer*

`www1`
`www2`
	`apache2`
	`php7.0`
	`php7.0-mysql`
		Installe le serveur Web Apache et ses extensions pour PHP7

`mysql`	`mysql-server`	*Installe le serveur de base de données MySQL*
`log`	`beep`	*Bip bip*

Synthèse 1 : Résumez en 4-6 lignes ce que vous avez fait depuis le début du TP.

Appelez votre chargé de TP pour lui faire une démonstration de la construction de votre VM.

4. Construction de la VM : aspects spécifiques

Dans les sous sections suivantes, vous trouverez des instructions spécifiques pour terminer la construction votre VM. Concentrez-vous sur votre section et ignorer les autres (elles seront réalisées par vos collègues et partagées avec Git).

4.1. Firewall

Le routage doit être activé* de manière persistante.

Le script d'activation du firewall (voir Tableau 1) doit être disponible dans le répertoire `/root` et avoir les droits d'exécution.

Vous devez donc modifier le contenu d'un fichier de manière automatisée (pas de manière interactive avec un éditeur de texte). Pour cela, nous vous conseillons d'utiliser une commande `sed` pour décommenter une ligne*, et d'utiliser un *provisionner* pour exécuter cette commande sur la VM.

4.2. Load balancer

La passerelle par défaut doit être configurée de manière persistante (il s'agit de `fw` …).

Le fichier de configuration du *load balancer* (voir Tableau 1) doit être placé dans le répertoire `/etc/haproxy`.

4.3. Serveurs Web

Le code PHP du serveur Web (voir Tableau 1) doit être placé dans le répertoire `/var/www/html`.

4.4. Serveur SQL

Le serveur SQL doit être autorisé à répondre aux requêtes venant de n'importe quel hôte client (par défaut, il ne répond que sur l'adresse de `loopback`).

Pour cela, vous devez commenter (avec un `#`) la ligne suivante dans `/etc/mysql/mariadb.conf.d/50-server.cnf` :

```
bind_address = 127.0.0.1
```

Nous vous conseillons d'utiliser une commande `sed` pour commenter une ligne*, et d'utiliser un *provisionner* pour exécuter cette commande sur la VM.

Enfin, il faut préparer la table SQL dans laquelle l'application Web va stocker les messages des utilisateurs. Le script SQL (voir Tableau 1) contient la configuration SQL nécessaire.

La commande suivante permet d'injecter cette configuration dans le serveur :

```
mysql -u root < app.sql
```

Utilisez un *provisionner* pour l'exécuter sur la VM.

Remarque : cette commande suppose que le script SQL existe sur la VM et se trouve dans le répertoire courant …

Le script SQL doit être effacé de la VM après l'initialisation de la table SQL.

4.5. Collecteur

Le script de supervision doit être placé dans le répertoire `/root` et avoir les droits d'exécution.

Synthèse 2 : Résumez en 4-6 lignes ce que vous avez fait depuis la dernière synthèse.

Appelez votre chargé de TP pour lui faire une démonstration de la construction de votre VM.

Woo Hoo! Votre infrastructure est prête …
mais il vous faut encore la tester (et surtout la débugger). *D'oh!*

5. Recette

Il est temps de passer à la deuxième phase du projet, illustrée dans la Fig. 4.

Clonez le dépôt Git dans un autre répertoire et lancez la construction de l'infrastructure virtuelle :

```
cd app
./build.sh
```

Si tout s'est bien déroulé, les six VM doivent être apparues dans la GUI de `VirtualBox`.

Démarrez-les et … depuis un autre PC, faites pointer le navigateur sur le *load balancer* :

```
http://198.51.100.2/index.php
```

Conseils :

Le PC *client* ne connait pas le réseau IP du *load balancer* (`198.51.100.0/24`). Il faut donc le lui apprendre, en ajoutant une route* pour ce réseau, passant par l'adresse IP *dynamique* de `fw`.

N'oubliez pas d'ajoutez une exception* pour ce réseau, dans la configuration du proxy.

Si la page principale de votre site apparait, c'est gagné ! Tapez quelques messages dans la zone de texte et vérifiez qu'ils sont bien affichés.

Si l'application ne fonctionne pas correctement, collaborez avec vos collègues pour troubleshooter et trouver l'origine du problème, puis faites-le corriger par le responsable de l'élément fautif.

Synthèse 3 : Expliquez en 4-6 lignes l'intérêt d'automatiser la création de VM.

Appelez votre chargé de TP pour lui faire une démonstration de la construction de l'application.

6. Mémos

6.1. Git

Cloner le dépôt `perlimpinpin.git` depuis le serveur `198.51.100.42` :

```
git clone 198.51.100.42:perlimpinpin.git
```

L'ensemble des fichiers du dépôt est téléchargé et placé dans le dossier `perlimpinpin`, créé automatiquement dans le dossier courant.

Personnaliser la configuration :

```
git config --global --edit
```

Modifier notamment les paramètres `name` et `email`.

Ajouter le fichier `proxy/proxy.json` au dépôt :

Se placer dans le répertoire du dépôt local, puis :

```
git add proxy/proxy.json
git commit proxy/proxy.json
git push origin master
```

Faire un `pull` sur le dépôt du serveur :

```
git pull
```

Récupère toutes les mises à jour `push`ées par les autres utilisateurs depuis votre dernier `pull`.

Uploader (*push*) vos contributions vers le dépôt du serveur :

Commencez par un *pull* si vous ne l'avez pas fait depuis un moment : entretemps, un collaborateur a peut-être fait un *push*, et vous devez nécessairement mettre à jour votre dépôt local avant d'envoyer vos contributions.

Ensuite :

```
git commit -a
```

Un éditeur de texte s'ouvre : vous devez alors résumer (en français) les modifications que vous avez apportées depuis votre dernier `push`.

Cette étape est importante, surtout lorsque vous travaillez en groupe sur les mêmes fichiers : il permet de garder une trace des contributions, de savoir qui a fait quoi et surtout pour quelle raison !

Enfin, synchroniser vos contributions avec le dépôt distant :

```
git push
```

Afficher toutes les contributions du dépôt :

```
git log
```

Inclut les contributions de tous les collaborateurs.

Documentation Git :

Pour les curieux :

https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git

6.2. Packer

Construire une VM :

`Packer` prend en entrée un fichier de configuration au format JSON, qui contient toutes les instructions de construction.

```
# Vérifier le fichier de conf
packer validate exemple/dns.json

# Construire une VM en utilisant les instructions
# du fichier de conf
packer build exemple/dns.json
```

Syntaxe du fichier de configuration :

Un fichier de configuration `Packer` doit contenir au moins :

- Un *builder* (directive `builder`*).

  Il est responsable de la création de la VM (à partir d'une image de base) et de son exportation. Il existe des *builders* pour `VirtualBox`, pour `VMware Workstation`, pour `Amazon EC2`, etc.

- Un ou plusieurs *provisionner(s)* (directive `provisionners`*).

  Chaque *provisionner* permet, par exemple, de copier un fichier ou d'exécuter une commande sur la VM

Ces informations permettent maintenant de comprendre le rôle du fichier `exemple/dns.json` :

Il se base sur une image OVA `debian-stretch.ova` située dans le répertoire personnel d'`etudiant` et dont le login est `root` / `vitrygtr`. Il s'en sert pour construire une VM `VirtualBox` nommée `DNS-GAU-FR-1` et équipée de deux cartes réseau. La seconde carte est en mode `Réseau interne` et branchée à un switch virtuel nommé `vSwitch42`.

Le provisionnement de la VM consiste à installer le paquetage `bind9`, copier quatre fichiers de configuration, et enfin exécuter le script `install.sh`.

A la fin de la construction de la VM, sa première carte réseau sera configurée en mode `Bridge` sur la carte physique `eth0`.

Syntaxe du *builder* pour `VirtualBox` :

Informations requises par le builder :

Propriété	Rôle
`type`	`virtualbox-ovf` indique que l'on veut construire une VM à partir d'une image OVA fournie
`source_path`	Le chemin de l'image OVA servant de base; peut être relatif (par rapport à l'emplacement du fichier JSON) ou absolu
`ssh_username` `ssh_password`	Les identifiants root de l'image de base
`vm_name`	Le nom de la VM à construire
`vboxmanage`	Une liste de commandes `VBoxManage*` exécutées au début de la construction de la VM, pour la paramétrer
`vboxmanage_post`	Identique à `vboxmanage`, mais les commandes sont exécutées à la fin de la construction de la VM

Liste des *provisionners* utiles :

Provisionner	Rôle
`file`	Copier un fichier depuis l'hôte vers la VM
`shell`	Exécuter, soit une (ou plusieurs) commandes sur la VM (directive `inline`), soit un script présent sur l'hôte (directive `script`). Dans ce dernier cas, le script est d'abord copié sur la VM, rendu exécutable, puis exécuté

Documentation `Packer` :

Pour les curieux :

https://www.packer.io/docs/builders/virtualbox-ovf.html
https://www.packer.io/intro/getting-started/build-image.html

6.3. *VirtualBox*

Créer un OVA compatible `Packer` :

Le *builder* `VirtualBox` peut utiliser toute image OVA comme base, à condition qu'elle respecte les caractéristiques suivantes :

- OVA version 2.0
- Une carte réseau en mode NAT
- Utilitaire `sudo` installé (si la VM est sous Linux)
- Login et mot de passe connus
- Serveur SSH installé, SSH avec le login `root` autorisé

Bien sûr, la VM fournie possède ces caractéristiques …

Configurer une VM en CLI :

La commande `VBoxManage` permet de réaliser toutes les manipulations sans passer par l'interface graphique : par exemple, ajouter de la RAM ou un disque dur, allumer la VM, … ou encore gérer les cartes réseau virtuelles.

```
# Ajouter une seconde carte réseau branchée sur
# le switch vSwitch42 (créé automatiquement)
VBoxManage modifyvm DNS-GAU-FR-1 --nic2 intnet
VBoxManage modifyvm DNS-GAU-FR-1 --intnet2 vSwitch42

# modifyvm ne peut être invoqué que si la VM est éteinte
VBoxManage modifyvm DNS-GAU-FR-1 --nic1 bridged
VBoxManage modifyvm DNS-GAU-FR-1 --bridgeadapter1 eth0

# Si la VM est allumée, il faut utiliser controlvm :
# Carte 1 configurée en Bridge sur en0
VBoxManage controlvm DNS-GAU-FR-1 nic1 bridged en0
# Carte 2 configurée en Réseau interne sur vSwitch42
VBoxManage controlvm DNS-GAU-FR-1 nic2 intnet vSwitch42
```

Pour invoquer `VBoxManage` dans `Packer`, il faut passer par les directives `vboxmanage` et `vboxmanage_post` du *builder* de VirtualBox. Les paramètres doivent être passés sous la forme d'un tableau (indiqué par les crochets `[ ]`) de chaines de caractères.

Exemple pour la première commande précédente (tirée du fichier `exemple/dns.json`) :

```
[
        "modifyvm",
        "{{.Name}}",
        "--nic2",
        "intnet"
]
```

`{{.Name}}` est une variable que `Packer` remplacera automatiquement par le nom de la VM.

Documentation `VirtualBox` :

Pour les curieux :

https://www.virtualbox.org/manual/ch08.html

6.4. Divers

Renommer un PC (ancien nom : `www2`, nouveau nom : `web2`)

En trois étapes :

1. Changer le nom :

```
hostnamectl set-hostname web2
```

2. Éditer le fichier `/etc/hosts` et remplacer les deux occurrences de `www2` par `web2` (deuxième ligne) :

```
127.0.0.1       localhost
127.0.1.1       www2.localdomain       www2
...
```

3. Fermer le terminal et le rouvrir. Observer le prompt du terminal* pour confirmer que les modifications ont été prises en compte.

Remarque : seul le `root` a le droit d'effectuer ces actions …

Configurer `eth0` en adressage statique persistant (adresse `203.0.113.10/24`) :

Modifier `/etc/network/interfaces` :

```
auto eth0
iface eth0 inet static
     address 203.0.113.10/24
```

Activer le routage :

Décommenter la ligne suivante dans `/etc/sysctl.conf` :

```
#net.ipv4.ip_forward=1
```

Le changement prendra effet au prochain démarrage.

Décommenter la ligne `#bind_address = 127.0.0.1` dans le fichier `/etc/mysql/my.cnf` :

```
# Voir UE3.1 Administration système pour les détails !
sed -E -i.SAVE \
     's/^#bind_address/bind_address/' \
     /etc/mysql/my.cnf
```

Commenter la ligne `net.ipv4.ip_forward=1` dans le fichier `/etc/sysctl.conf` :

```
# Voir UE3.1 Administration système pour les détails !
sed -E -i.SAVE \
     's/^net.ipv4.ip_forward/#net.ipv4.ip_forward/' \
     /etc/sysctl.conf
```