«DOCKER & CISCO ACI»

Thèse de Bachelor présentée par

Monsieur Vincent UDRIOT

pour l'obtention du titre Bachelor of Science HES-SO en

Ingénierie des technologies de l'information avec orientation en Communications, multimédia et réseaux

Septembre 2016

Professeur HES responsable

Gérald Litzistorf

En collaboration avec

Cisco Suisse

LeShop.ch - Migros

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Automne 2016
Session de bachelor

INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION

ORIENTATION COMMUNICATIONS, MULTIMEDIA ET RESEAUX

DOCKER & CISCO ACI

Descriptif :

Docker constitue une évolution intéressante dans le data center en cloisonnant des services sur un système Linux amenant ainsi une nouvelle manière de concevoir les applications auparavant monolithiques. Les annonces complémentaires ne manquent pas comme son support par Microsoft dans Server 2016.

En complément à l'étanchéité garantie par les containers, ce travail doit proposer des solutions réseaux appropriées (VLAN, firewall, …) pour isoler les flux de données

Il doit par ailleurs montrer l'intérêt du couplage avec la solution Cisco ACI – Nexus 9000

Travail demandé :

Ce travail comprend les étapes suivantes :

1) Etude de Docker
   - Introduire les principaux concepts théoriques
   - Les expliquer à partir d'exemples pédagogiques
   - Traiter en détail des parties réseau, cluster et stockage
   - Montrer les avantages & inconvénients de cette architecture dans des scénarios utilisés par LeShop
   - Comparer avec les solutions basées sur VMware ESXi

2) Etude de Cisco ACI (Application Centric Infrastructure)
   - Introduire les principaux concepts théoriques
   - Les expliquer à partir d'exemples pédagogiques avec des termes simples et précis
   - Respecter le cahier des charges proposé par Cisco
   - Définir les tests unitaires
   - Quelles sont les options possibles au niveau du design ?
   - Présenter les avantages et inconvénients

3) Synthèse rencontrées
   - Mentionner les difficultés
   - Peut-on conserver les bonnes pratiques du monde physique ?
   - Doit-on réfléchir autrement ?
   - Analyse sur la sécurité

Sous réserve de modification en cours du travail de Bachelor

Candidat :
M Vincent Udriot
Filière d'études : ITI

Professeur(s) responsable(s) :
Litzistorf Gérald

En collaboration avec : Cisco & LeShop
Travail de bachelor soumis à une convention de stage en entreprise : non
Travail de bachelor soumis à un contrat de confidentialité : non

Timbre de la direction

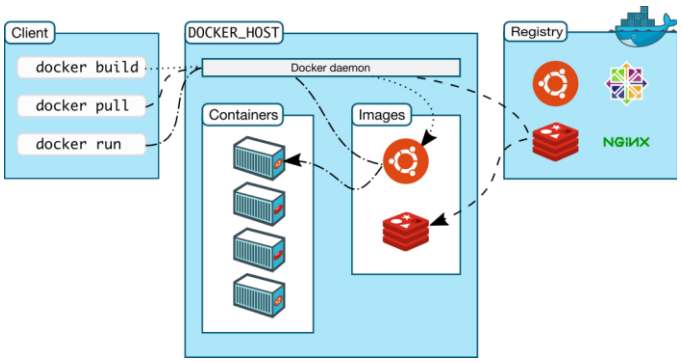
h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Automne 2016
Session de bachelor

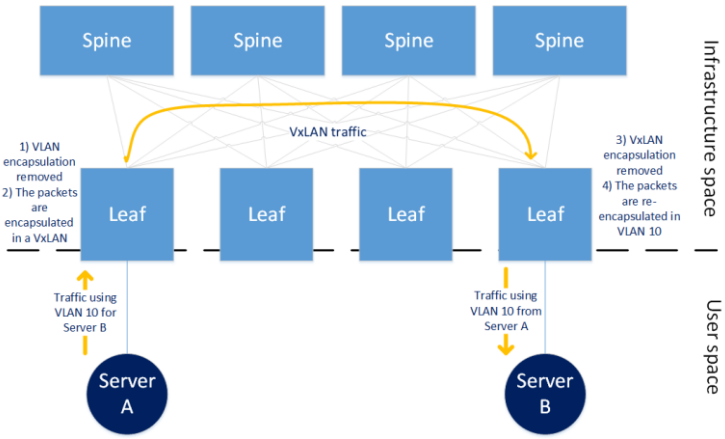
Résumé :

Ce travail de bachelor est une étude du fonctionnement de Docker et une étude de la solution réseau Cisco ACI qui sont tous deux au cœur des dernières avancées technologiques dans le monde du datacenter.

Dans une première phase, il propose une introduction aux microservices et plus particulièrement à Docker ainsi que ses outils avec différents exemples d'implémentation et une analyse des différentes fonctionnalités : réseaux, volumes, etc.

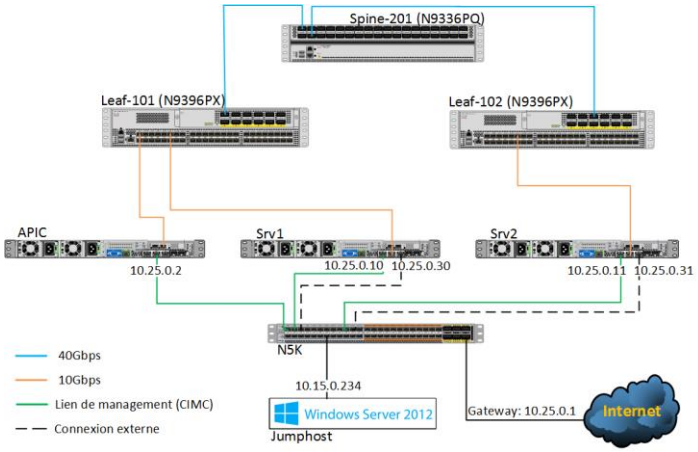


La création d'un petit cluster avec Docker Swarm illustre les possibilités qu'offrent les containers au sein d'un cloud.



Dans une deuxième phase, ce travail aborde le fonctionnement de l'infrastructure réseau Cisco ACI avec son approche particulière centrée sur l'application et les concepts théoriques qui y sont afférents.

Finalement, ce document aborde le projet Contiv qui permet de lier Docker avec Cisco ACI tout en analysant le bénéfice qu'apporte un couplage de ces deux solutions ainsi que leurs avantages et inconvénients.



Candidat :
M Vincent Udriot
Filière d'études : ITI

Professeur(s) responsable(s) :
Litzistorf Gérald

En collaboration avec : Cisco & LeShop
Travail de bachelor soumis à une convention de stage en entreprise : non
Travail de bachelor soumis à un contrat de confidentialité : non

Timbre de la direction

Remerciements

Je tiens à remercier Monsieur Gérald Litzistorf, mon professeur HES responsable du projet, pour son suivi tout au long de ce travail, pour ses remarques avisées et pour m'avoir mis en contact avec Cisco et LeShop.

Merci à Monsieur Bremtane Moudjeb de Cisco Rolle pour son soutien, ses conseils et pour m'avoir mis en contact avec l'équipe de Cisco Zürich.

Je souhaite également remercier MM. Rolf Schaerer, Jonas Walker et Dang Ngo de Cisco Zürich pour l'infrastructure qu'ils m'ont mise à disposition ainsi que pour leur appui technique dans le couplage d'ACI avec Docker.

De plus, je remercie Sebastien Pasche et Benoît Chalut de LeShop.ch (Migros) pour avoir partagé avec moi leur expérience et leurs conseils avisés au sujet de Docker.

Finalement, j'adresse mes remerciements à mes parents et à mes amis pour leur soutien inconditionnel.

Table des matières

1 Docker

1.1 Principe de microservice

De nos jours, l'informatique au sein des datacenters a pris des proportions qu'il n'aurait pas été possible de prévoir il y a 15 ans. Nous sommes passé d'une vue monolithique des applications où les programmes avec toutes leurs fonctions tournaient sur une machine à une nouvelle approche qui consiste en un découpage des applications en microservices destinés à être lancés sur différentes machines. Chaque morceau contient un seul processus et a un but unique.

Cette approche a fait évoluer la manière dont les équipes sont construites chez les développeurs. Désormais, elles sont construites autour des fonctions de l'application, chaque équipe travaille sur un processus.
De cette manière, la maintenance, la mise-à-jour ou la correction d'une application est facilitée, car il s'agit de se concentrer sur des processus précis au lieu de voir le tout comme un monolithe. Pour finir, la scalabilité se conçoit de manière plus efficiente, il n'est pas nécessaire de mettre à disposition d'avantage de ressources à toute l'application si c'est seulement un de ses processus qui a des besoins particuliers.

1.2 Introduction à Docker

Docker est un projet Open Source de containers qui apportent, par définition, une couche d'abstraction permettant la portabilité des applications tout en assurant une certaine isolation, le tout restant très léger. C'est pour cela que cette solution s'inscrit dans le monde des microservices.

Le projet, lancé en 2013, a pour but de simplifier le cycle de vie des applications développées en permettant de garder le contrôle sur les dépendances de celles-ci et donc d'en rendre la mise en production plus facile et plus rapide (portabilité) tout en permettant de les isoler du système d'exploitation qui les exécute et des autres applications (sécurité) grâce au principe de container.

Docker est tout particulièrement adapté à l'utilisation dans le Cloud, il intègre d'ailleurs nativement des outils pour faire tourner des containers facilement et rapidement sur les plateformes des leaders du cloud tels que Google[1], Amazon ou encore Microsoft Azure.

Le projet est accompagné par de nombreux contributeurs et une communauté non-négligeable. Selon le site OpenHub[2], 619 personnes ont contribué au projet sur GitHub au cours des 12 derniers mois. En outre, on compte parmi les entreprises qui ont contribué au projet des sociétés telles que RedHat, IBM, Google ou encore Cisco[3] :

```
Top changeset contributors by employer
(Unknown)                4520 (47.9%)
"Docker"                 3821 (40.5%)
"Red Hat"                 685 (7.3%)
"IBM"                     232 (2.5%)
"Google"                  119 (1.3%)
"Cisco"                    49 (0.5%)
"Amadeus"                   4 (0.0%)
"VMWare"                    2 (0.0%)
"CoreOS"                    1 (0.0%)
```

[1] https://cloud.google.com/container-engine/
[2] https://www.openhub.net/p/docker
[3] Chiffres tirés de l'utilitaire Gitdm : https://gist.github.com/arun-gupta/7d5a373099ff831d7213

Docker articule même un total de plus de 1500 contributeurs.[4]

A la base, supporté uniquement sur des machines Linux, Microsoft a fait un partenariat avec Docker afin de créer une version compatible avec son système d'exploitation. Cette nouveauté prouve que la containerisation a de beaux jours devant elle. Avec Windows Server 2016, il est désormais possible d'utiliser cette mouture nativement supportée[5].

[4] Source : https://docker.wistia.com/medias/fqwm0x9tgz
[5] https://www.docker.com/microsoft

1.3 Comparaison avec les machines virtuelles

Les containers sont souvent comparés avec les machines virtuelles classiques, car ils sont en de nombreux points identiques, mais sont, sur d'autres points bien différents.

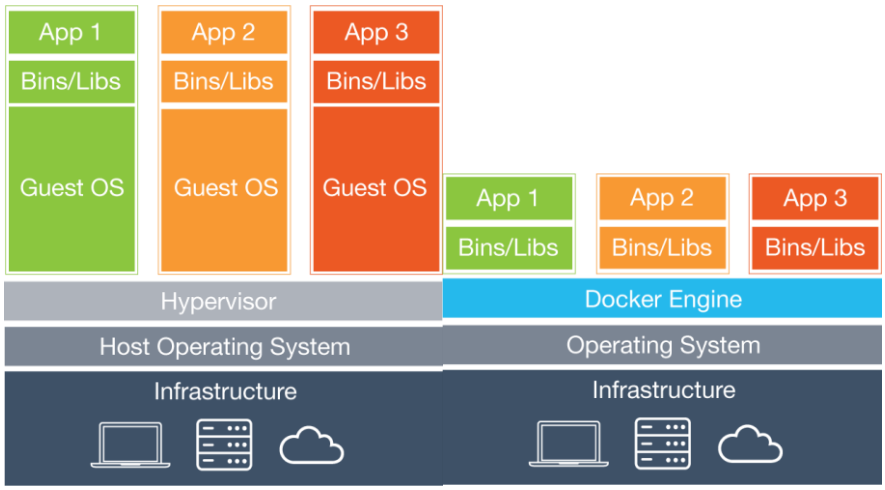


Figure 1 Architecture d'une vm (gauche) et d'un container (droite)[6]

Leur principale différence se situe dans leur architecture respective. En effet, les applications dans le monde Docker sont lancées par le système d'exploitation de l'hôte dont elles partagent une partie du kernel tout en restant indépendantes par rapport à l'infrastructure sous-jacente à condition de ne pas leur mettre des points d'ancrage avec leur système hôte (montage du système de fichier local par exemple).
Docker se passe donc de l'hyperviseur classique en le remplaçant par le « Docker Engine » et il se passe également du système d'exploitation invité, puisque celui-ci n'est plus nécessaire, tout en gardant les dépendance (librairies et binaires) nécessaires.
Par conséquent, l'architecture de Docker nécessite moins de ressources (mémoire vive et espace disque) et donc offre une grande portabilité des applications, mais ne permet pas de faire tourner des applications compilées pour d'autres systèmes d'exploitation que celui de la machine hôte.

En outre, au jour d'aujourd'hui, la maturité des machines virtuelles n'est pas négligeable par rapport à celle des containers Docker qui en sont encore à leurs débuts. En effet, bien que les technologies utilisées par les 2 types de systèmes existent depuis de nombreuses années, VMware, par exemple, a été créée en 1999 alors que Docker est seulement apparu en 2013. En revanche, les bases techniques des containers existent depuis bien longtemps, déjà en 1979 l'appel *chroot* qui a pour but de changer le répertoire racine d'un processus apporte une première forme d'isolation.

1.4 Principaux composants de Docker

Docker peut être découpé en 3 principales entités :

1) Client
2) Daemon
3) Registry

[6] Source : https://www.docker.com/what-docker

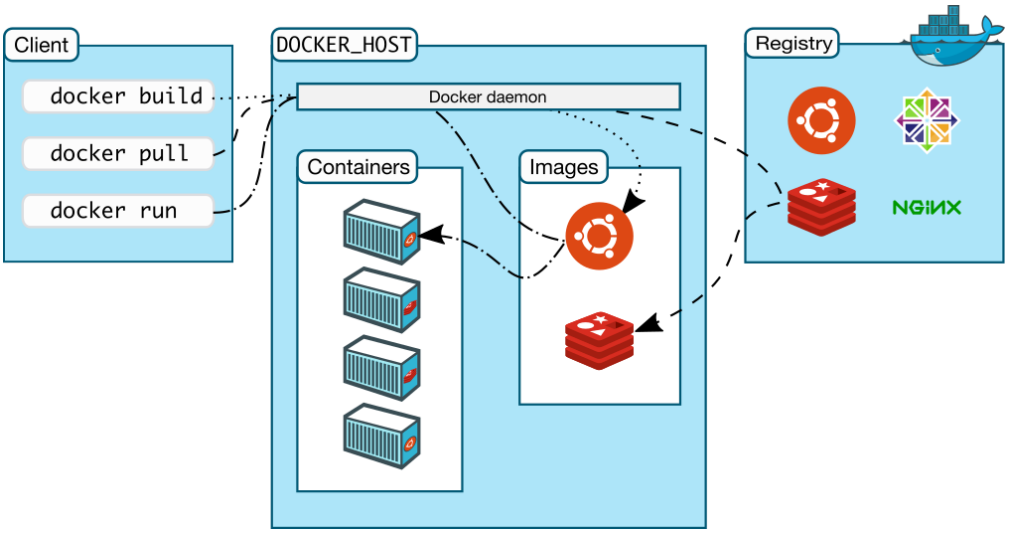


Figure 2 Les principaux composants de Docker

1.4.1 Le client

Le rôle du client est simple, il permet à l'administrateur d'envoyer des commandes au daemon grâce à la ligne de commande[7]. De plus, Docker intègre une API REST qui permet aux utilisateurs de créer des outils adaptés à leurs besoins et ainsi mettre en place un système automatisé. Le site officiel en fournit une documentation très bien construite[8].

1.4.2 Le daemon

Le cœur de Docker réside dans le daemon, c'est lui qui gère les containers en fonction des commandes envoyées par le client. Il peut créer des containers en se basant sur des images locales ou étant hébergées dans un registry, démarrer, arrêter, supprimer des containers, créer des images etc.

1.4.3 Le registry

Les containers étant créés à partir d'images, il est nécessaire de les stocker afin d'y faire appel quand c'est nécessaire. C'est le rôle du registry.
Le projet Docker met à disposition un registre officiel qui contient des centaines d'images de containers contenant des applications répandues telles que Nginx, Busybox, MySQL, MongoDB, Java, Python etc.

1.5 Installation

Afin de pouvoir utiliser Docker, il faut installer *docker-engine*[9] qui est au cœur de la solution de container.
Cette petite procédure a pour objectif de montrer comment installer Docker sur une machine Linux

[7] https://docs.docker.com/engine/reference/commandline
[8] https://docs.docker.com/engine/reference/api/docker_remote_api
[9] https://docs.docker.com/engine/installation

sans interface graphique, mais il faut savoir qu'il est également possible de disposer d'un GUI fonctionnant sur Windows[10] ou sur Mac OS X[11] grâce à Docker Toolbox.

L'installation ci-dessous a été effectuée sur une machine virtuelle tournant sur Fedora Server 23 64 bit (Kernel 4.2.3) afin d'obtenir les paquets les plus récents. La machine virtuelle a été installée sur VirtualBox pour des raisons de flexibilité et de portabilité.

Prérequis :

- Linux 64-bit
- Kernel 3.10 au minimum
- Paquets à jour

Dans une première phase, il faut ajouter le dépôt Yum de Docker :

```
tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/fedora/$releasever/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
```

et installer le paquet docker-engine :

```
dnf install docker-engine
```

Il est, dès lors, possible de démarrer le daemon de Docker :

```
systemctl start docker
```

1.5.1 Lancement d'un container basique

Afin de tester que l'installation a bien fonctionné, on peut essayer de lancer un container simple et de voir l'effet de cette commande.

```
docker run -i -t ubuntu /bin/bash
```

Paramètres :

- *run* crée et lance un nouveau container
- *-i* garde l'entrée standard ouverte (interactif)
- *-t* image alloue un pseudo-tty et indique l'image à utiliser pour créer le container
- *Commande* à lancer dans le container une fois celui-ci démarré

[10] https://docs.docker.com/engine/installation/windows/
[11] https://docs.docker.com/engine/installation/mac/

Effets sur le système :

1) Le moteur Docker vérifie que l'image fournie en paramètre existe localement, dans le cas contraire, il va la chercher sur Docker Hub, le registry officiel. (**Pull**)
2) Crée le container à partir de l'image.
3) Allocation d'un système de fichier et ajoute la couche lecture-écriture permettant à l'application d'effectuer ses opérations.
4) Création d'une interface réseau donnant la possibilité au container de dialoguer avec l'hôte.[12]
5) Affectation d'une adresse IP
6) Lance l'application
7) Attache le container à la console (entrée-sortie-erreur standards)

1.6 Images Docker

Les images sont un élément clé de Docker, c'est elles qui sont à l'origine des containers, c'est donc elles qui sont utilisées lorsque l'on veut pouvoir lancer un container sur une autre machine, on peut les comparer à un plan de construction.
A ce titre, il est intéressant de voir comment créer ou mettre à jour une image afin, par exemple, de l'adapter à nos besoins.

1.6.1 Création d'une image à partir d'un container

Pour illustrer l'importance de la mise à jour d'une image, j'ai choisi un exemple simple : imaginons que nous utilisons l'image de base « ubuntu », mais elle manque de la commande *ping* dont nous avons besoin.

Dans un premier temps, pour ce scénario, nous allons lancer le container en arrière-plan (détaché grâce à l'option *-d*) en lui donnant un nom (option *--name*) pour montrer comment on entre dans un container qui est déjà lancé

```
docker run -d --name my_ubuntu -t ubuntu /bin/bash
```

On entre ensuite dans le container, ce qui peut être fait de deux manières :

1) Grâce à la commande *docker attach* qui fonctionnera si un script est en cours d'exécution dans le container :

```
docker attach my ubuntu
```

2) Avec la commande *docker exec*, l'option *–i* (interactif) et la commande à exécuter dans le container, ce qui fonctionnera toujours :

```
docker exec -i -t my_ubuntu bash
```

Maintenant que nous sommes dans le container, nous allons installer la commande *ping* qui fait partie du paquet *inetutils-ping* et finalement, après l'installation, quitter le container :

[12] Les détails de la partie réseau sont abordés plus loin dans ce document

```
apt-get update
apt-get install -y inetutils-ping
exit
```

Pour finir, il ne reste plus qu'à créer une image à partir du container qu'il soit arrêté ou non (l'image sera plus vite créée si le container est à l'arrêt) :

```
docker commit my_ubuntu my_new_image
```

L'utilisateur dispose désormais d'une nouvelle image prenant plus d'espace que celle d'origine. Il peut désormais démarrer de nouveaux containers et se basant sur celle-ci.

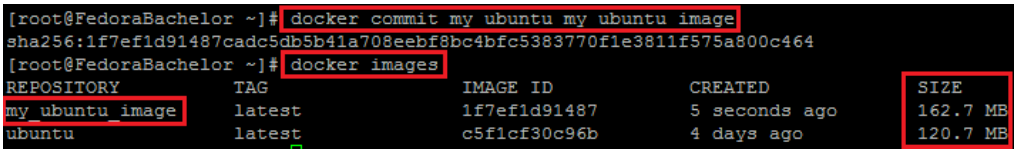


Figure 3 Création d'une image à partir d'un container

1.6.2 Les dockerfiles

Au lieu de créer un container à partir d'une image, de le modifier et d'en faire une nouvelle image, Docker permet de créer une image à partir d'une autre image en utilisant un dockerfile qui pourrait être comparé à une recette de cuisine.

Un dockerfile est un fichier texte qui ressemble à un script tout simple dont les instructions commencent par des mots-clés très bien présentés dans la documentation officielle[13].

Le scénario suivant en propose un exemple basique.

1.6.3 Scénario : Serveur Nginx à contenu statique

Dans ce scénario, je vais présenter de quelle manière il est possible de créer une image personnalisée d'un serveur Nginx qui mettra à disposition, sur le réseau local, les fichiers Html de mon choix.

Je me base sur une image de container Nginx mise à disposition par le repository officiel de Docker (Docker Hub[14]).

Dans un premier temps, il faut créer le dossier dans lequel se trouveront les différents « ingrédients » nécessaires à savoir : le dockerfile qui indiquera les étapes de la création de l'image et le dossier, nommé « staticContent », contenant les fichiers Html qui seront distribués par Nginx.

[13] https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#the-dockerfile-instructions
[14] https://hub.docker.com/explore

Dans ce cas, le dockerfile est très simple :

```
FROM nginx
COPY staticContent /usr/share/nginx/html
```

Figure 4 Fichier firstNginx.df

L'instruction *FROM* indique l'image de base à utiliser et *COPY* demande à Docker de copier le dossier source de l'hôte (chemin relatif par rapport au dockerfile) vers le dossier destination du container.

L'image peut désormais être construite :

```
docker build -t first_nginx_img -f firstNginx.df .
```

L'option *–t* permet de « tager » l'image, ce tag est utilisé ici pour donner un nom à l'image, l'option *-f* indique le nom du dockerfile et le paramètre final indique le dossier dans lequel se trouvent les fichiers pour la création de l'image (En l'occurrence, le dossier courant).

Si le paramètre *–f* n'est pas spécifié, le nom cherché par défaut est *dockerfile*.

Attention : Docker n'autorise par les majuscules pour le nom de l'image, cela cause une erreur dont le message n'indique pas clairement l'origine.

```
[root@localhost dockerTests]# docker build -t first_nginx_img -f firstNginx.df .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM nginx
latest: Pulling from library/nginx

efd26ecc9548: Pull complete
a3ed95caeb02: Pull complete
a48df1751a97: Pull complete
8ddc2d7beb91: Pull complete
Digest: sha256:2ca2638e55319b7bc0c7d028209ea69b1368e95b01383e66dfe7e4f43780926d
Status: Downloaded newer image for nginx:latest
 ---> 6f8d099c3adc
Step 2 : COPY staticContent /usr/share/nginx/html
 ---> 2d8c6e932937
Removing intermediate container af00c98eb1ec
Successfully built 2d8c6e932937
```

Figure 5 Résultat du build avec un dockerfile

Lors de l'exécution, on observe 2 étapes : 1) Docker charge l'image de base à savoir Nginx. 2) Il effectue la copie du dossier contenant les fichiers Html dans la nouvelle image.
Durant ce processus, Docker passe par un container temporaire et lorsque les modifications nécessaires sont effectuées, il le supprime.

Il ne reste plus qu'à créer un container basé sur cette image en spécifiant le nom désiré.

```
docker run --name nginx-server -d -p 80:80 first_nginx_img
```

Le paramètre *–p <port de l'hôte> <port du container>* mappe des ports de l'hôte avec des ports du container[15] rendant ainsi le serveur accessible dans le réseau grâce à l'adresse web http://adresse-IP-du-host.

[15] https://docs.docker.com/engine/userguide/networking/default_network/binding

Dans ce scénario, l'administrateur du serveur web est obligé de regénérer l'image et de recréer le container s'il veut compléter ou mettre à jour ses pages web puisque le dossier contenant les fichiers Html est copié dans l'image lors du build.

1.7 Les volumes

Bien souvent, les processus exécutés produisent ou nécessitent des fichiers créés par d'autres containers ou par l'administrateur, il est donc nécessaire de pouvoir y accéder facilement tout en maximisant l'isolation que permet le container.
En séparant les données des containers, l'administrateur assure la persistance des données. Si le container est supprimé, le volume et son contenu ne disparaissent pas.

Afin de détacher les données d'un container et afin de les rendre accessibles à d'autres containers, Docker a créé le concept de volumes[16].
Un volume est en fait un dossier (ou dans certains cas un simple fichier) qui est monté dans le container à l'emplacement choisi par l'utilisateur.
Par défaut, le montage est fait en lecture/écriture, mais il peut être fait en lecture uniquement, ce qui peut être intéressant pour protéger les données.

Ainsi, si on reprend l'exemple du serveur Nginx, on peut faire du dossier */usr/share/nginx/html* un volume pour que l'administrateur puisse directement modifier ses fichiers html sans avoir à reconstruire l'image du container.

```
docker run --name secondNginx -d -v /usr/share/nginx/html nginx
```

Grâce à l'outil *docker inspect*, il est possible de trouver le point de montage (en rouge) :

```
docker inspect secondNginx
```

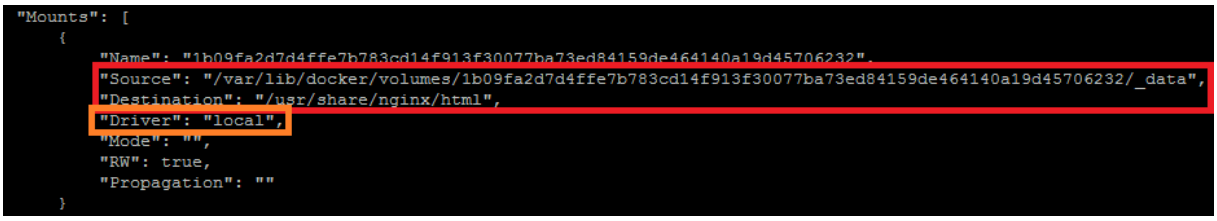


Figure 6 Résultat de la création d'un volume local

Le driver indiqué en orange (*local*) nous indique que ce point de montage est sur la machine hôte, mais il est également possible de monter des volumes distants grâce à des plugins pour Docker tels que Convoy, Flocker, GlusterFS et bien d'autres[17]. C'est d'ailleurs nécessaire d'utiliser des volumes indépendants de l'hôte afin d'assurer que le container soit déplaçable.

Cette méthode peut s'avérer bien utile dans le cas où plusieurs containers, répartis sur différents hôtes, doivent accéder à des fichiers communs pour effectuer une tâche bien définie comme le présente Michael Crosby sur son site.[18]
Dans son cas, 3 containers sont utilisés pour différents processus : un serveur de jeu (Minecraft), un

[16] https://docs.docker.com/engine/userguide/containers/dockervolumes
[17] https://docs.docker.com/engine/extend/plugins
[18] http://crosbymichael.com/advanced-docker-volumes.html

générateur de carte de type Google Maps et un serveur web pour afficher la carte du jeu aux personnes intéressées de suivre l'évolution du serveur de jeu

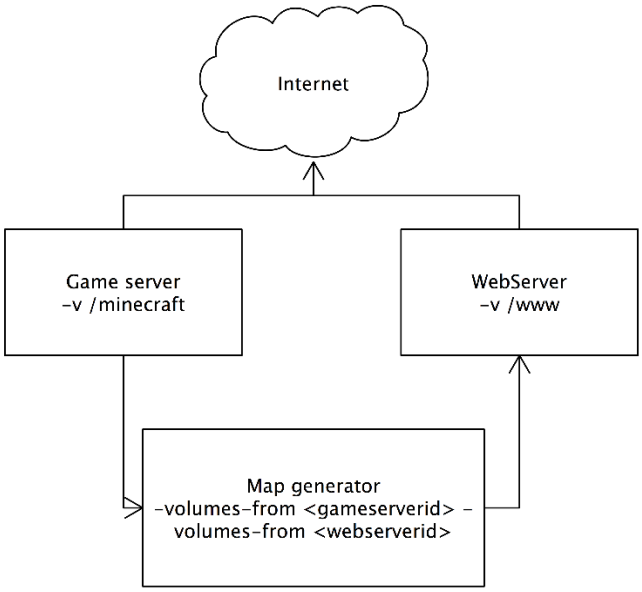


Figure 7 Exemple de volumes partagés par plusieurs containers

Ainsi la carte que le jeu crée peut être analysée par le générateur de carte topographique qui fournira à son tour un résultat que le serveur web mettra à disposition des visiteurs.
Comme l'illustre la figure, 2 volumes sont nécessaires pour faire ceci : */minecraft* et */www* créés avec, respectivement, GameServer et WebServer. Ils sont rattachés à MapGenerator grâce au paramètre « *--volumes-from* ».

1.7.1 Monter des volumes NFS

Dans une optique de cluster, il est capital de pouvoir partager des volumes entre containers qui se trouvent sur des machines différentes, mais il est aussi important de disposer de nœuds spécialisés dans le stockage simplifiant, ainsi, la gestion de l'espace disque et les backups tout en optimisant les performances.

A l'heure actuelle, Docker, à lui seul, ne permet pas de monter des volumes distants, seuls les volumes locaux peuvent être créés. Pour monter et gérer des volumes distants, il faut faire appel à des plugins officiels ou indépendants pour Docker tels que docker-volume-netshare[19] que je vais utiliser pour présenter la manière de procéder.
La procédure d'installation de ce dernier n'étant pas forcément facile à aborder, il est intéressant de s'y attarder.

[19] https://github.com/gondor/docker-volume-netshare

1.7.1.1 Installation de docker-volume-netshare

Prérequis :

La procédure ci-dessous est réalisée sur un hôte utilisant Fedora Server 23. Le paquet « nfs-utils » est donc considéré comme étant déjà installé.

Démarche :

Ce plugin étant écrit en Go et cette procédure passant par la compilation des sources, il est nécessaire de télécharger les paquets liés à ce langage ainsi que les paquets pour git :

```
dnf install go git
```

Cette opération nécessite également de renseigner la variable d'environnement *$GOPATH*. Pour ce faire, il faut ajouter les lignes suivantes au fichier ~/.bashrc :

```
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

Et recharger le fichier avec :

```
source ~/.bashrc
```

La machine a désormais tous les composants nécessaires à l'obtention, la compilation et l'installation du plugin : (Le dossier courant est /root)

```
go get github.com/gondor/docker-volume-netshare
```

Il faut ensuite intégrer ce service à systemd et l'ajouter aux services lancés au démarrage :

```
touch /etc/systemd/system/docker-volume-netshare.service
chmod 664 /etc/systemd/system/docker-volume-netshare.service

tee /etc/systemd/system/docker-volume-netshare.service <<-'EOF'
[Unit]
Description=Docker-volume-netshare daemon
After=network.target

[Service]
ExecStart=/root/go/bin/docker-volume-netshare nfs

[Install]
WantedBy=default.target
EOF

systemctl daemon-reload
systemctl enable docker-volume-netshare.service
systemctl start docker-volume-netshare.service
```

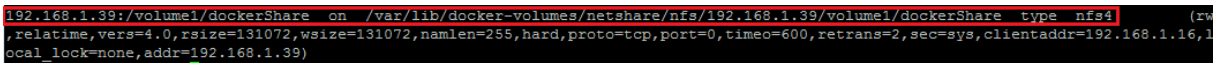
Il est désormais possible de créer un volume Docker qui sera situé sur un volume NFS :

```
docker run -d --name my_nginx --volume-driver=nfs -v
192.168.1.39/volume1/dockerShare:/usr/share/nginx/html -p 80:80 nginx
```

Comme le montre la capture ci-dessous, Docker a créé un dossier source sur lequel il a monté le volume NFS, ce que confirme la commande *mount*.

```
"Mounts": [
    {
        "Name": "192.168.1.39/volume1/dockerShare",
        "Source": "/var/lib/docker-volumes/netshare/nfs/192.168.1.39/volume1/dockerShare",
        "Destination": "/usr/share/nginx/html",
        "Driver": "nfs",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```


Figure 8 Résultat de la création d'un volume distant (montage NFS)



```
192.168.1.39:/volume1/dockerShare on /var/lib/docker-volumes/netshare/nfs/192.168.1.39/volume1/dockerShare type nfs4 (rw
,relatime,vers=4.0,rsize=131072,wsize=131072,namlen=255,hard,proto=tcp,port=0,timeo=600,retrans=2,sec=sys,clientaddr=192.168.1.16,l
ocal_lock=none,addr=192.168.1.39)
```


Figure 9 La commande mount montre le volume distant que le driver a monté

1.8 Cluster

Un des principaux avantages de Docker est la facilité et la rapidité de déploiement des containers indépendamment de l'hôte sur lequel ils sont déployés. Cette abstraction du matériel permet dès lors de créer un cluster d'hôtes pouvant accueillir les nombreux containers qu'une entreprise voudrait déployer.

Dans ce chapitre, je vais présenter plus précisément Swarm[20], la solution native de clustering proposée par Docker, en abordant son architecture ainsi que la manière de la mettre en place dans un contexte simple.
Le choix de Docker Swarm pour ce chapitre est purement arbitraire, il existe d'autres solutions telles que Google Kubernetes ou Apache Mesos pour les plus connue. La comparaison de ces différents systèmes pourrait tout à faire l'objet d'un projet à part entière au vu de son étendue, c'est pour cela que j'ai choisi de n'en aborder qu'un.

1.8.1 L'architecture de Docker Swarm

L'architecture de Swarm est principalement composée de 4 types d'éléments :

1) Un client
2) Un manager (ou 1 primary et des secondaries si utilisés en haute-disponibilité)
3) Un key-value store
4) Des nœuds pour les containers

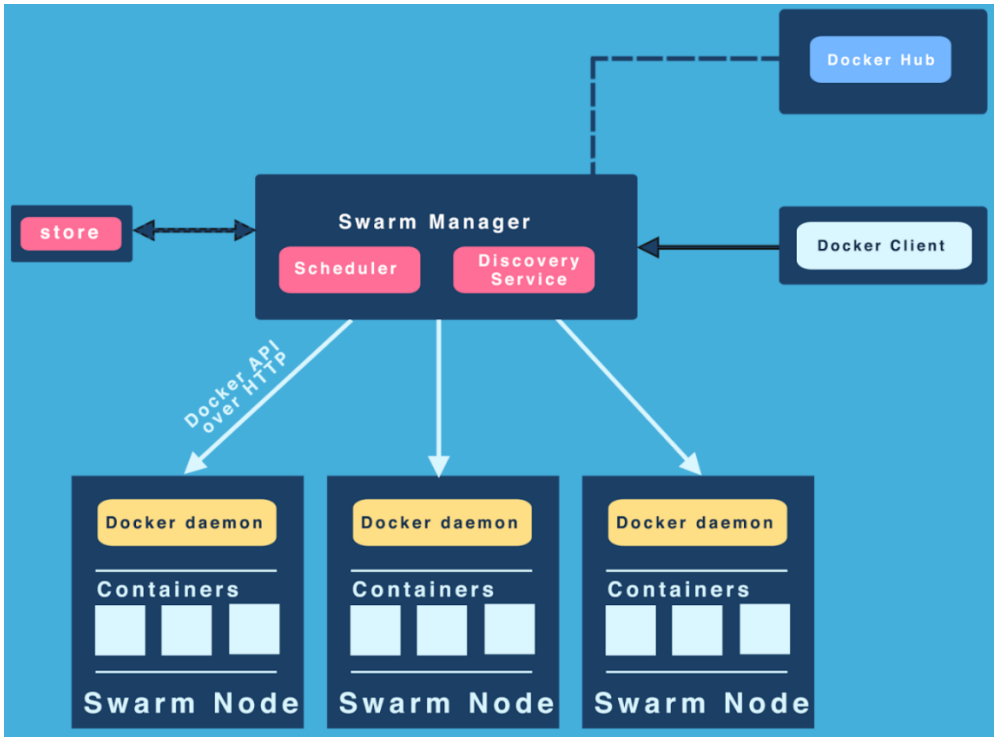


Figure 10 Infrastructure de clustering Docker Swarm[21]

[20] https://docs.docker.com/swarm/overview
[21] https://learning-continuous-deployment.github.io/docker/conclusion/2015/06/14/conclusion/

1) **Le client**
   Du point de vue de l'utilisateur d'un cluster fonctionnant sur Swarm, rien ne change par rapport à une infrastructure à une seule machine. En effet, il exécute les mêmes commandes à destination du manager que sur un hôte.
2) **Le key-value store**
   Ce service est une sorte de base de données qui recense les participants au cluster et leur rôle ainsi que leur « état de santé » afin de pouvoir, en tout temps, fournir au manager une liste des nœuds exploitables.
   Différents key-value store sont supportés par Docker tels que Consul, etcd ou encore Zookeeper[22].
3) **Le manager**
   Le travail du manager peut être divisé en 2 rôles distincts : l'ordonnanceur et le service de découverte.
   L'ordonnanceur se charge de choisir les nœuds sur lesquels lancer les containers en fonction de différents paramètres (Filtres, stratégies et contraintes). Le service de découverte, quant à lui, dialogue avec le key-value store afin de maintenir à jour la liste des machines à disposition pour l'exécution de ses tâches.
4) **Les nœuds**
   Tous équipés du *docker-engine* et d'un container *Swarm*, Ils exécutent les ordres du manager et font tourner les containers.

Alexandre Beslic, un ingénieur de Docker Swarm, présente les principaux concepts avec d'avantage de détails dans une série de vidéos officielles de bonne qualité disponibles sur Youtube : https://www.youtube.com/watch?v=sg9wNcdMhbU

[22] https://docs.docker.com/swarm/discovery

1.8.2 Exemple d'implémentation de Swarm

La mise en place d'un cluster *Swarm* n'est pas forcément facile à effectuer, car la documentation officielle[23] est très dense et les exemples présentés comprennent d'avantage d'outils que nécessaire, ce qui a tendance à rendre leur compréhension pénible.

Le but de cet exemple d'implémentation est donc d'être le plus simple et le plus accessible possible. Il est implémenté sur 4 machines virtuelles faisant tourner Fedora 23 server sur lesquels est installé Docker.

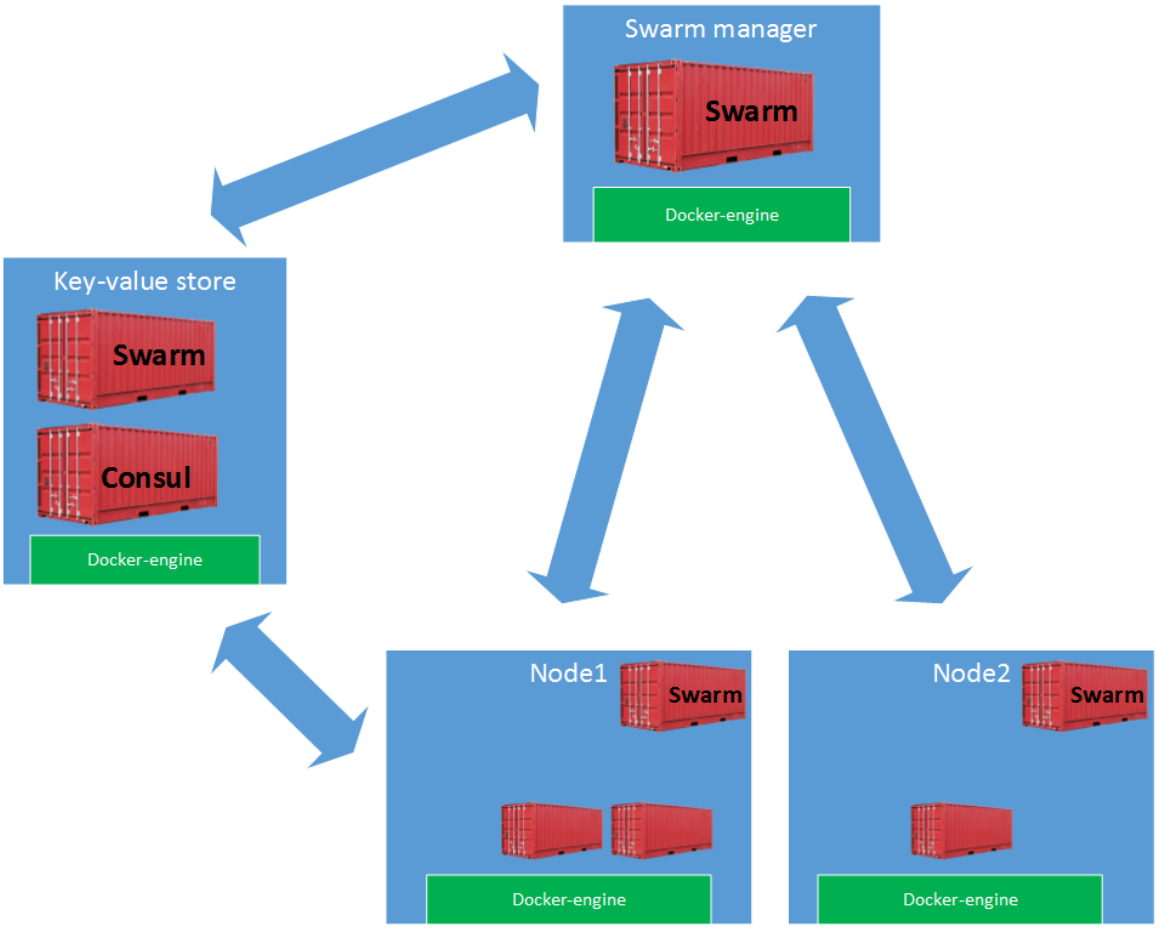


Figure 11 Schéma de l'exemple d'implémentation de Swarm

[23] https://docs.docker.com/swarm

Démarche :

Afin de suivre cette procédure sans encombre, la première phase consiste à ouvrir les ports nécessaires dans le firewall des machines. En effet, ce point m'a fait perdre beaucoup de temps, car les erreurs obtenues n'étaient pas explicites. De plus, le firewall de Fedora est préconfiguré de sorte que l'ajout d'une règle à la suite des règles existantes ne sert à rien, il faut donc à tout prix indiquer que la règle ajoutée doit prendre la première place. Docker utilisant iptables pour effectuer certaines tâches, il serait problématique de le désactiver ou de le flusher.

Le manager doit pouvoir recevoir les commandes sur le port 4000 ainsi que les connexions des participants au cluster sur le port 2375 :

```
iptables -I INPUT 1 -p tcp --dport 4000 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 2375 -j ACCEPT
```

La machine exécutant le key-value store doit pouvoir recevoir des connexions provenant du manager ainsi que des connexions à destination de Consul :

```
iptables -I INPUT 1 -p tcp --dport 2375 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 8500 -j ACCEPT
```

Les nœuds doivent, quant à eux, pouvoir recevoir les paquets provenant du manager :

```
iptables -I INPUT 1 -p tcp --dport 2375 -j ACCEPT
```

Une fois les firewalls correctement configurés, il faut lancer le service Docker en arrière-plan sur chaque machine en le mettant en écoute sur le port 2375 :

```
docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock &
```

Sur la machine du key-value store, il faut lancer le container de Consul qui est en écoute sur le port 8500 :

```
docker run -d -p 8500:8500 --name=consul progrium/consul -server -bootstrap
```

Ensuite, passons au lancement du manager (*manager_ip* et *consul_ip* doivent être remplacés par les adresses respectives des deux machines). L'option *--advertise* indique à Swarm qui est le manager et qui est le key-value store :

```
docker run -d -p 4000:4000 swarm manage -H :4000 --advertise \
<manager_ip>:4000 consul://<consul_ip>:8500
```

Pour finir, les machines node0 et node1 doivent s'annoncer auprès du key-value store :

```
docker run -d swarm join --advertise=<node_ip>:2375 consul://<consul_ip>:8500
```

Résultat :

Grâce à la commande *docker –H :4000 info*, j'obtiens une vue d'ensemble de mon cluster avec, notamment, le nombre de containers, leur état, le nombre de nœuds, leur statut, leur version etc.

```
[root@FedoraBachelor-2 ~]# docker -H :4000 info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 9
Server Version: swarm/1.2.1
Role: primary
Strategy: spread
Filters: health, port, containerslots, dependency, affinity, constraint
Nodes: 2
 FedoraBachelor-4: 192.168.1.17:2375
  └ ID: OIHU:IYJU:7EQM:MNDI:UPH6:2GKC:NYIV:LPF6:QUWY:NEPC:DCBU:466X
  └ Status: Healthy
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.018 GiB
  └ Labels: executiondriver=, kernelversion=4.4.8-300.fc23.x86_64, operatingsyst
em=Fedora 23 (Twenty Three), storagedriver=devicemapper
  └ Error: (none)
  └ UpdatedAt: 2016-05-09T08:40:25Z
  └ ServerVersion: 1.11.1
 FedoraBachelor-5: 192.168.1.18:2375
  └ ID: TGN2:FF7H:ARH5:YJGY:IHDY:UFWW:4LYH:SFDD:PMOG:JWEB:JMJA:PJMF
  └ Status: Healthy
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.018 GiB
  └ Labels: executiondriver=, kernelversion=4.4.8-300.fc23.x86_64, operatingsyst
em=Fedora 23 (Twenty Three), storagedriver=devicemapper
  └ Error: (none)
  └ UpdatedAt: 2016-05-09T08:40:23Z
  └ ServerVersion: 1.11.1
Plugins:
 Volume:
 Network:
Kernel Version: 4.2.3-300.fc23.x86_64
Operating System: linux
Architecture: amd64
CPUs: 2
Total Memory: 2.037 GiB
Name: 591c92139005
Docker Root Dir:
Debug mode (client): false
Debug mode (server): false
```

Figure 12 Etat du cluster Swarm

Pour lancer un container sur le cluster, il suffit d'utiliser la commande docker avec l'option *–H :4000* (H signifiant host) comme ceci :

```
docker -H :4000 run busybox /bin/sh
```

La commande suivante permettra alors de voir quelle machine a lancé le container :

```
docker -H :4000 ps -a
```

1.8.3 La politique d'ordonnancement de Swarm

Tout l'intérêt de disposer d'un cluster est de pouvoir répartir la charge sur les différentes machines le composant grâce à une politique d'ordonnancement et à des règles automatiques ou définies par l'administrateur.

Les stratégies[24] définissent de quelle manière sont distribués les containers (en remplissant machine après machine, en distribuant de manière aléatoire ou en distribuant en fonction de la charge actuelle des nœuds).

Swarm travaille également avec toute une série de filtres[25] permettant d'établir des règles très précises comme le fait que 2 containers d'un certain type ne peuvent pas se trouver sur la même machine, la disponibilité d'un port etc.

1.9 Réseau

Docker offre un grand nombre de possibilités quand il s'agit de réseau, le but de ce chapitre est d'en donner un aperçu. Pour aller un peu plus dans le détail, le projet LorisPack a créé un excellent résumé de ce qu'offre Docker en matière de réseau[26].

1.9.1 Réseaux de base

A la base, Docker propose 3 configurations possibles[27] :

1) Bridge
2) None
3) Host

1.9.1.1 Bridge docker0

Le bridge est la solution choisie par défaut. Chaque container a une interface qui est reliée à une interface virtuelle et qui va elle-même être connectée à un bridge virtuel qui est finalement relié à l'interface de l'hôte. Lorsqu'un container est démarré sans spécification particulière concernant le réseau (port mapping grâce à l'option *--expose* ou *–p* de *docker run*), il n'a pas accès au réseau physique, mais il peut, en revanche, joindre les autres containers qui sont lancés à ses côtés. Ce type de réseau est amplement suffisant pour une utilisation simple de containers.

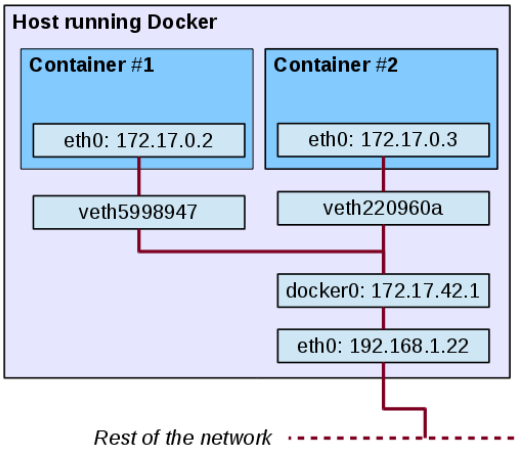


Figure 13 Configuration réseau par défaut[28]

Le lien 25 aborde en détails le fonctionnement de ce type de réseau.

[24] https://docs.docker.com/swarm/scheduler/strategy
[25] https://docs.docker.com/swarm/scheduler/filter
[26] http://fr.slideshare.net/lorispack/docker-networking-101
[27] https://docs.docker.com/engine/userguide/networking
[28] Source : https://www.linuxjournal.com/content/concerning-containers-connections-docker-networking

1.9.1.2 None

Comme son nom l'indique, none connecte le container à aucun réseau, ce qui est très peu utilisé, mais qui peut être intéressant si le but est d'isoler totalement une application.

1.9.1.3 Host

Le réseau « Host » place le container directement derrière l'interface réseau de l'hôte comme s'il était l'hôte lui-même. En d'autres termes, l'isolation du réseau existante dans les autres modes est absente. Le container partage le *namespace* réseau de la machine hôte, ce qui, en pratique, veut dire qu'il a la même adresse IP que son hôte, qu'il a accès à toutes les interfaces physiques et qu'il est, par conséquent, directement exposé au réseau.

Il n'est donc pas nécessaire de mapper des ports entre le container et l'hôte, mais le container n'est plus isolé avec tout ce que cela implique en termes de sécurité.
En pratique, ce mode est surtout utilisé lorsque les besoins en performance sont élevés. Grâce à lui, on s'affranchit donc d'une surcouche qui nécessiterait des ressources.

Lancé avec le mode par défaut (bridge docker0), le container ne voit que l'interface de son bridge :

```
[root@FedoraBachelor-3 test-compose]# docker run -it --name debianTest debian:jessie  bash
root@ab597b30156f:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:4/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 14 Bridge docker0

alors qu'avec le mode host, il a directement accès à toutes les interfaces :

```
[root@FedoraBachelor-3 test-compose]# docker run -it --name debianTest --net=host debian:jessie  bash
root@FedoraBachelor-3:/# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:83:62:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.14/24 brd 192.168.1.255 scope global dynamic enp0s3
       valid_lft 231848sec preferred_lft 231848sec
    inet6 fe80::a00:27ff:fe83:6223/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:82:a2:2d:f0 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:82ff:fea2:2df0/64 scope link
       valid_lft forever preferred_lft forever
5: vetha7909ba@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether de:ec:b9:54:a8:0a brd ff:ff:ff:ff:ff:ff
    inet6 fe80::dcec:b9ff:fe54:a80a/64 scope link
       valid_lft forever preferred_lft forever
7: veth9b90b9f@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 8e:16:6d:dc:14:05 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8c16:6dff:fedc:1405/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 15 Mode host

1.9.2 User-defined network

Pour des utilisations plus avancées, il est possible de configurer le réseau en fonction de ses besoins grâce à la commande *docker network*. A ce titre, il existe 2 types de réseaux définis par l'utilisateur :

1) Bridge network
2) Overlay network

1.9.2.1 Bridge network

En créant ses propres bridges et en connectant ses containers comme bon lui semble (sur plusieurs réseaux internes à la fois, par exemple), l'administrateur d'un système basé sur Docker dispose d'une grande flexibilité.

La commande clé qu'il utilisera pour créer son propre bridge est :

```
docker network create --driver bridge my_bridge
```

```
[root@FedoraBachelor ~]# docker network ls
NETWORK ID          NAME                DRIVER
fd42cc9604bd        bridge              bridge
d0614f6707d6        docker_gwbridge     bridge
8959ea32e221        host                host
f123b3456f9a        my_bridge           bridge
91bbdbb2a6be        none                null
```

Figure 16 Le nouveau bridge apparaît dans les réseaux Docker à disposition

```
[root@FedoraBachelor ~]# docker network inspect my_bridge
[
    {
        "Name": "my_bridge",
        "Id": "f123b3456f9af85f108c6e042fdd42d9900ca687d5565b95ff14ffff959fcf1e",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Figure 17 La configuration du bridge et de son sous-réseau

Il est dès lors possible d'y attacher un container lors de sa création simplement grâce à l'option *--net* :

```
docker run -d --net=my_bridge busybox
```

La seule restriction qu'impose le driver *bridge* est que tous les containers que l'on veut mettre sur un réseau commun doivent être sur le même hôte, ce qui n'est pas nécessairement le cas dans un cluster.

1.9.2.2 Overlay network

J'en arrive donc aux réseaux multi-hôtes[29]. Un des aspects proposés par Docker est la possibilité de créer un réseau entre containers se trouvant sur différents hôtes grâce au driver *overlay*.

Pour assurer cette fonction tout en maintenant l'isolation qu'est censé proposer tout réseau, Docker s'appuie sur la technologies VxLAN (Virtual Extended Local Area Network standardisé en 2014 dans la RFC 7348[30]) dont le but est identique à celui des VLAN traditionnels, mais le type d'encapsulation est différent.
Alors qu'il est possible d'avoir 4096 VLAN ids différents, VxLAN propose plus de 16.7 millions de VxLAN ids. En revanche, ces « VLAN étendus » provoquent un overhead non-négligeable par rapport aux VLAN classiques à cause de la méthode d'encapsulation (environ 50 octets).

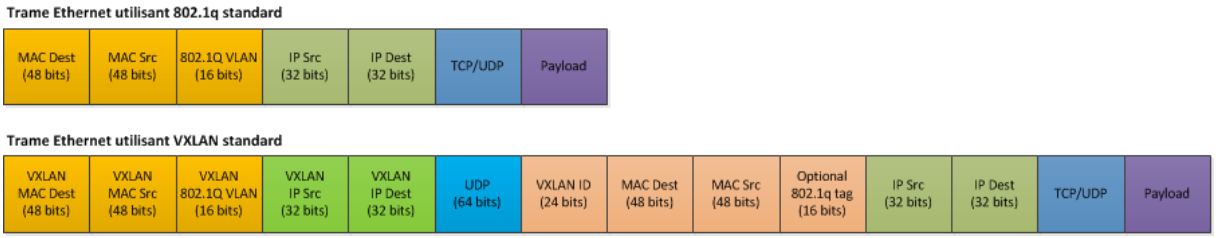


Figure 18 Trame VLAN vs VxLAN[31]

Ce format d'encapsulation a principalement été mis au point par VMware, Cisco et RedHat.
Une bonne introduction à ce protocole réalisée par David Mahler, un technical leader de Blue Box (une société d'IBM) est visible sur Youtube[32].

Afin d'illustrer cette possibilité, j'ai décidé de mettre en place un petit scénario impliquant 3 containers sur différents hôtes.
Pour ce faire, je me base sur la procédure de création du cluster détaillée dans le chapitre précédent et j'y ajoute un hôte, car l'*overlay network* nécessite, tout comme pour la mise en place d'un cluster, un key-value store pour le service de *discovery*.

Deux changements doivent cependant être effectués :
Premièrement, nous devons ouvrir les ports pour Serf (une solution de gestion des membres de clusters[33]) et VxLAN sur chaque machine en plus des ports déjà ouverts dans la procédure de création de cluster :

```
iptables -I INPUT 1 -p tcp --dport 7946 -j ACCEPT
iptables -I INPUT 1 -p udp --dport 4789 -j ACCEPT
```

Deuxièmement, le daemon doit être lancé avec deux options supplémentaires afin qu'il obtienne toutes les informations nécessaires auprès du key-value store, ce qui donne cette commande :

```
docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --cluster-
advertise=<ip_de la machine>:2375 --cluster-store=consul://<ip consul>:8500 &
```

[29] https://docs.docker.com/engine/userguide/networking/get-started-overlay
[30] https://www.rfc-editor.org/rfc/rfc7348.txt
[31] Source : https://www.randco.fr/blog/2013/vxlan
[32] https://www.youtube.com/watch?v=Jqm_4TMmQz8
[33] https://www.serfdom.io/

Tout le reste est identique, il ne reste donc plus qu'à se servir du Swarm manager afin de propager un nouveau réseau :

```
docker -H :4000 network create --driver overlay --subnet=10.0.9.0/24 my-net
```

En se rendant, dès lors, sur n'importe quelle machine, on peut s'apercevoir que le réseau est présent :

```
[root@FedoraBachelor-5 ~]# docker network ls
NETWORK ID          NAME                DRIVER
496dccaac0fa        bridge              bridge
d2caf1a976e8        host                host
68a2a898560a        my-net              overlay
8c8947ce3203        none                null
```

Afin d'analyser ce qu'il se passe sur le réseau, il faut créer deux containers sur le cluster (ici un Nginx et un Ubuntu de base) en s'assurant qu'ils soient bien sur 2 hôtes différents grâce à une contrainte :

```
docker -H :4000 run -dt --net=my-net --name ubuntu --
env="constraint:node==FedoraBachelor-5" ubuntu

docker -H :4000 run -dt --net=my-net --name nginx3 --
env="constraint:node==FedoraBachelor-4" nginx
```

Le container Ubuntu n'étant pas équipé de la commande *ping*, je l'ai directement ajoutée en me connectant au nœud qui l'héberge :

```
docker exec ubuntu apt-get update
docker exec ubuntu apt-get install -y inetutils-ping
```

```
[root@FedoraBachelor-2 ~]# docker -H :4000 inspect nginx3 | grep IP
            "IP": "192.168.1.18",
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "",
            "IPPrefixLen": 0,
            "IPv6Gateway": "",
                    "IPAMConfig": null,
                    "IPAddress": "10.0.9.4",
                    "IPPrefixLen": 24,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
```

Figure 19 IP obtenue par le container Nginx3

Grâce au fichier généré par TCPDUMP que j'ai ouvert à l'aide de Wireshark, j'ai pu ainsi analyser les échanges VxLAN capturés :

```
tcpdump -w capture -i enp0s3 &
docker exec ubuntu ping 10.0.9.4
```

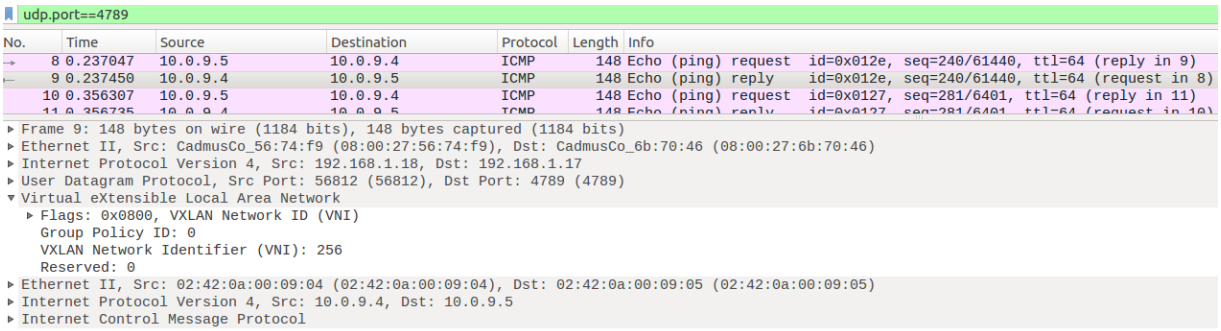


Figure 20 Capture de VxLAN

Sur cette capture, l'encapsulation de la couche 2 sur UDP grâce à VxLAN est visible et il est également possible d'observer que le VxLAN id est 256.

Ce type de réseau avancé est présenté plus en détails dans une très bonne vidéo de la DockerCon2015 : https://www.youtube.com/watch?v=vb7U_9AO7Ww

1.10 Docker-Compose

Comme le principe de Docker est justement de décomposer les applications en différents processus qui peuvent ensuite être dimensionnés en fonction des besoins et éventuellement répartis sur différentes machines d'un cluster, il nous faut un moyen simple de définir une application ainsi que les différents containers qui la compose. Le nom est donc tout trouvé : Docker-Compose[34].

Le fonctionnement de cet outil est assez simple, il se base sur un fichier yml qui définit les différents containers et leurs liens réseau. Lorsque la commande *docker-compose up -d* est lancée, ce fichier est lu, les containers sont préparés puis lancés.

1.10.1 Exemple simple de compose

Pour illustrer ceci, je vais créer une application toute simple composée de deux containers : l'un exécutant redis (un système de base de données nosql) et l'autre faisant tourner une petit API Python qui va incrémenter un petit compteur dans la base de données et en afficher le résultat.

Cet exemple est inspiré d'une série de tutoriels[35] créés par un ingénieur de l'équipe de socketplane[36].

Dans une première phase, il faut installer Docker-compose, ce qui est rapide puisque nous avons déjà ajouté les repositories de Docker à notre machine puis nous aller créer un dossier de travail :

```
dnf install docker-compose
mkdir test-compose && cd test-compose
```

[34] https://docs.docker.com/compose/
[35] https://gist.github.com/botchagalupe/53695f50eebbd3eaa9aa
[36] http://socketplane.io

Dans ce dossier, il faut, ensuite, ajouter tous les fichiers qui permettront à notre application de fonctionner.
Tout d'abord, le code Python en lui-même qui est une API REST utilisant Flask qui va simplement se connecter à la base de données redis, puis incrémenter un compteur nommé « hits » et, pour finir, retourner une phrase avec le résultat du compteur.

```
tee app.py <<-'EOF'
from flask import Flask
from redis import Redis
import os
app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' %
redis.get('hits')

if   name   == "  main  ":
    app.run(host="0.0.0.0", debug=True)
EOF
```

L'application étant composée d'un container faisant tourner le code Python, il faut le définir grâce à un dockerfile. Il va se baser sur l'image officielle python :2.7, ajouter tous les fichiers du dossier courant à un dossier nommé « code » ajouté dans le container puis lancer depuis ce dossier la commande *pip* permettant l'installation de librairies pour Python. Le paramètre *-r* indique un fichier contenant la liste de celles dont l'application a besoin (en l'occurrence flask et redis)

```
tee dockerfile <<-'EOF'
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
EOF
```

```
tee requirements.txt <<-'EOF'
flask
redis
EOF
```

Une fois ces composants préparés, il ne reste plus qu'à créer le fichier yaml suivant :

```
tee docker-compose.yml <<-'EOF'
web:
  build: .
  command: python app.py
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
EOF
```

Il est divisé en 2 sections : la première concernant le container web et la deuxième concernant le container redis.
L'instruction *build* lance l'opération de création de l'image à partir du fichier *dockerfile* (nom par défaut). *Command* indique la commande que le container exécutera, *ports* aura l'effet équivalent au paramètre *-p* de docker run (mappage du port interne au container avec le port de la machine). *Volume* montera simplement le dossier *code* dans le container.
L'instruction la plus intéressante, car la plus puissante, est *links* qui va faire le lien entre les deux containers, ce qui nous évite d'avoir à indiquer le port utilisé.

Le container redis est simplement basé sur l'image officielle de redis à disposition sur Docker Hub.

L'ensemble des mots-clés pour le « compose file » sont à disposition sur le site officiel de Docker : https://docs.docker.com/compose/compose-file/

Il suffit finalement d'exécuter la commande *docker-compose up -d* dans le dossier courant, elle va, par défaut, faire appel au fichier *docker-compose.yml*, ce qui déclenchera le processus de création des containers.

La commande *docker-compose ps* permet de voir les containers lancés :

```
[root@FedoraBachelor-3 test-compose]# docker-compose ps
       Name                     Command               State          Ports
-------------------------------------------------------------------------------
testcompose_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
testcompose_web_1     python app.py                    Up      0.0.0.0:5000->5000/tcp
```

Figure 21 docker-compose ps montre le résultat

On peut alors constater que les deux containers ont un nom automatique basé sur le nom du dossier, le nom du container ainsi qu'un chiffre pour différencier les containers dans les cas où on choisit de lancer plusieurs fois le même container.

En cas de problème, il peut être très utile de faire appel à la commande *docker-compose logs* qui affiche la console des containers, permettant ainsi de cibler les erreurs.

L'application tourne correctement comme on peut le constater en appelant l'API Python avec cURL[37] :

```
[root@FedoraBachelor-3 test-compose]# curl 192.168.1.14:5000
Hello World! I have been seen 1 times.[root@FedoraBachelor-3 test-compose]# curl
 192.168.1.14:5000
Hello World! I have been seen 2 times.[root@FedoraBachelor-3 test-compose]# curl
 192.168.1.14:5000
Hello World! I have been seen 3 times.[root@FedoraBachelor-3 test-compose]#
```

Figure 22 L'application fonctionne avec 2 containers

1.10.2 Limitations Docker-Compose & Swarm

Il aurait été intéressant de pouvoir déployer cette application sur le cluster Swarm mis en place précédemment, mais malheureusement Swarm est, à ce jour, encore trop limité, l'ordonnancement sur plusieurs nœuds du cluster fonctionne mal, car il est aléatoire. Il n'est pas possible de prévoir quel container sera lancé en premier et, par conséquent, les dépendances entre eux (liens réseaux en l'occurrence) ne peuvent être assurées.
La seule solution est donc d'avoir recours à un ordonnancement manuel ce qui perd tout son sens puisque la force de Docker-Compose est justement l'automatisation.

De plus, un tel déploiement n'est possible que si le cluster est déployé à l'aide de Docker Machine (qui permet de gérer l'installation des Docker Engines sur différentes machines/ cloud providers), chose que je n'ai pas faite.

Pour obtenir d'autres résultats et contourner ces problèmes, il faudrait donc faire appel à un autre système de cluster tel que Kubernetes[38] dont l'inconvénient, en revanche, est d'utiliser une CLI, des définitions YAML et une API différentes que celles de Docker[39].

1.11 Conclusion

A travers cette première étape, j'ai pu découvrir le projet Docker que je ne connaissais que de nom, j'ai pu me rendre compte de sa puissance ainsi que tous les projets qui l'entourent et toutes les possibilités qu'ils apportent ensemble.

Si ce document avait abordé tous les composants officiels et toutes les possibilités de Docker tels que la restriction par l'administrateur des ressources dédiées à certains containers, le provisionnement d'un cluster à l'aide de *Docker-machine*[40] et bien d'autres, il aurait été beaucoup plus conséquent.

Il est certain que ce sujet s'inscrit complètement dans l'air du temps, une période où la *scalabilité* et le time-to-market sont des notions capitales. Il est d'ailleurs très motivant d'étudier des technologies qui sont au cœur de l'informatique de demain.

La suite de ce travail doit alors amener à comprendre ce qu'est Cisco ACI et ce qu'elle peut apporter à une infrastructure telle qu'un cloud Docker par exemple.

---

[37] cURL = Client URL Request Library, un outil simple permettant d'accéder à une ressource. e.g : une API REST
[38] http://kubernetes.io/
[39] Source : https://technologyconversations.com/2015/11/04/docker-clustering-tools-compared-kubernetes-vs-docker-swarm/
[40] https://docs.docker.com/machine/overview/

2 Cisco ACI

2.1 Quels sont les besoins actuels ?

De nos jours, l'informatique et l'industrie en général ont beaucoup évolué. Les technologies de l'information et de la communication se sont démocratisées et touchent un nombre élevé d'utilisateurs que ce soit des particuliers ou des entreprises.
Il y a 30 ans, seules les grandes entreprises avaient les moyens et le besoin d'avoir une infrastructure informatique (Serveurs, réseaux, etc.), mais, aujourd'hui, avec la démocratisation des technologies, la majorité des startups ou des particuliers veulent avoir accès à des services à bas prix sans avoir à attendre.
La notion de time-to-market devient alors cruciale, ce qui entraîne la nécessité de limiter le temps entre la conception d'un nouveau logiciel, d'un nouveau service informatique et sa mise en production.

Du point de vue des équipes informatiques, ce processus peut s'avérer long et fastidieux. En effet, l'informatique traditionnelle fonctionne « en silos » : l'équipe de développement met au point une application, puis se tourne vers les responsables bases de données pour obtenir ce dont ils ont besoin, il faut ensuite demander à l'équipe infrastructure de mettre à disposition le réseau et les machines nécessaires à la mise en production, consulter l'équipe sécurité, de stockage etc.

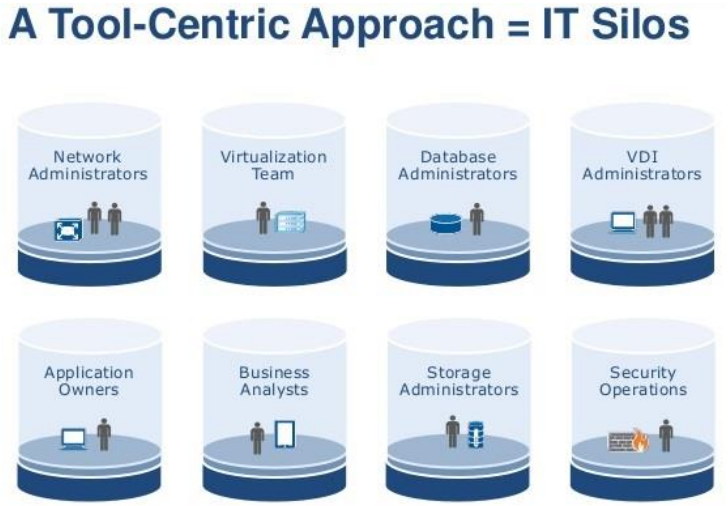


Figure 23 L'informatique traditionnelle "silotée"[41]

Chacun de ces silos est un monde à part qui a sa manière de fonctionner, son langage, etc., ce qui ne simplifie pas particulièrement la mise en place de nouveaux services et va donc à l'encontre des besoins du marché.

Un autre grand défi que l'IT doit relever est la gestion de systèmes très hétérogènes qui peuvent rendre difficile l'uniformisation des procédures, la résolution des problèmes et la gestion des ressources. En matière de cloud, l'idée de cloud hybride est apparue au cours de ces dernières années. Dans le but d'optimiser les coûts, il est, en effet, intéressant de mélanger son cloud privé avec les services proposés par d'autres entreprises au sein de leur cloud.
Il faut également compter sur le fait qu'aucun service provider n'a qu'un seul client, il est donc important, pour des raisons de sécurité (confidentialité, intégrité…), que l'infrastructure

[41] Source : http://fr.slideshare.net/ExtraHop/how-to-use-big-data-to-transform-it-operations

informatique rende possible l'isolation des différents tenants[42] (clients ou départements d'une entreprise). Pour ce faire, la solution généralement consiste, notamment, à utiliser des VLAN qui doivent être configurés et propagés. Ceci peut s'avérer long et représente une source potentielle d'erreur humaine non-négligeable qu'il est nécessaire de minimiser.

Au-delà de la rapidité et de la simplicité de déploiement d'un service, il ne faut pas non plus négliger la vitesse de localisation et de résolution des problèmes qui pourraient apparaître. Lorsqu'un client appelle pour se plaindre du fonctionnement d'une application, il faut pouvoir connaître sans délai l'origine du dérangement pour honorer le SLA et assurer au mieux la qualité des services.

Lorsque, finalement, un service arrive à son échéance ou à la fin de son cycle, l'infrastructure utilisée doit pouvoir être libérée rapidement afin d'être réutilisée pour un autre service/un autre client. Le temps pendant lequel des ressources sont mobilisées sans rapporter d'argent doit absolument être minimisé afin de limiter les pertes. Cette notion s'exprime donc par un besoin important d'automatisation

En conclusion, cette envie de mettre rapidement à disposition des services numériques tout en optimisant les coûts et en en assurant la qualité nous amène irrémédiablement à un besoin de faire évoluer la manière dont les datacenters sont conçus. C'est dans ce contexte que la solution « Application Centric Infrastructure » (ACI) de Cisco s'inscrit.

2.2 L'architecture Cisco ACI

ACI est une solution de provisionnement du réseau centré sur les applications contrairement à l'approche traditionnelle plutôt centrée sur le réseau en lui-même. Elle lie un aspect matériel et un aspect logiciel.

Elle est principalement composée de 3 parties : la Fabric, l'APIC (Application Policy Infrastructure Controller) et le reste des éléments du réseau.

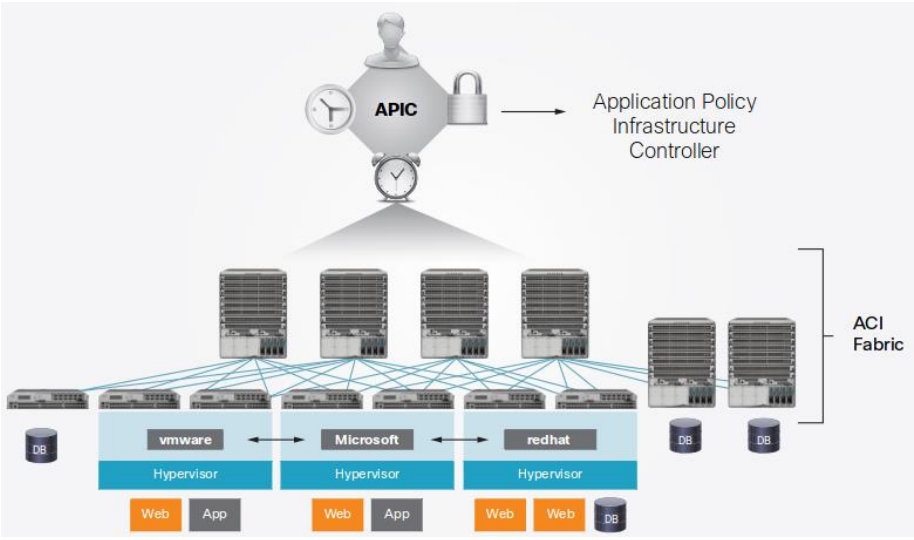


Figure 24 L'infrastructure type ACI[43]

[42] Tenant au sens anglais du terme signifie organisation, on parle de « multitenant infrastructure »
[43] Source : https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/aci-fabric-controller/at-a-glance-c45-729864.pdf

2.2.1 La Fabric

La fabric ACI représente le cœur du réseau physique et est construite selon une architecture « Spine - Leaf »[44].

2.2.1.1 Les origines de l'architecture spine-leaf

Traditionnellement, l'architecture qui prime dans les datacenters est l'architecture des « 3 tiers »[45] (Core, aggregation, access). Ce modèle a été créé pour répondre efficacement à la demande en termes de trafic « nord-sud », c'est-à-dire le trafic entrant et sortant du datacenter.
Comme en témoigne le schéma suivant, les liens dans ce modèle sont soumis au spanning-tree qui évite la formation de boucles dans le réseau tout en assurant que des liens soient redondants.

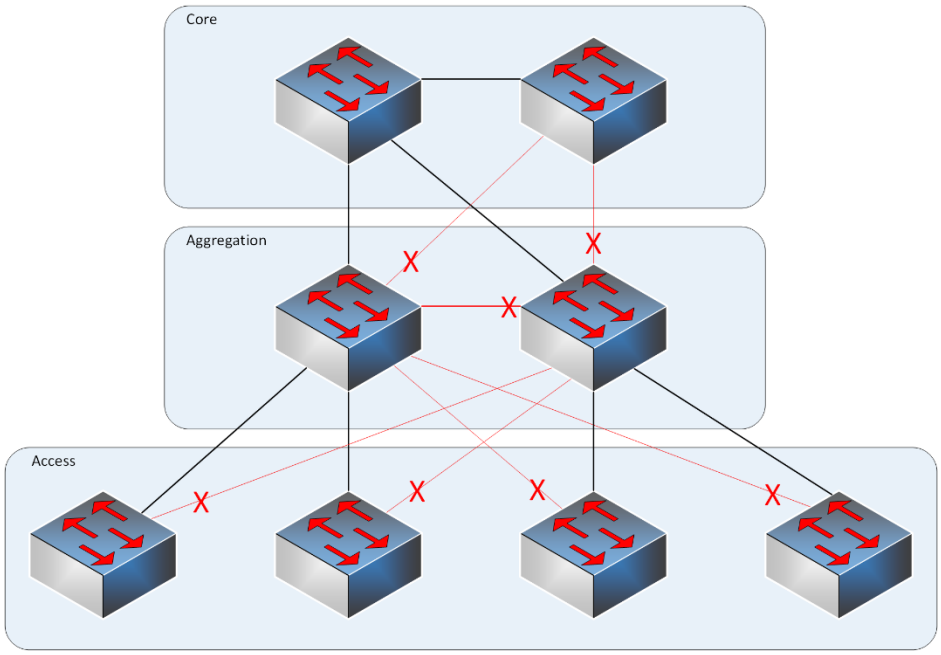


Figure 25 Le modèle "3-tiers"[46]

Avec le temps, les charges réseau ont évolué, en effet, on assiste de plus en plus à une augmentation du trafic est-ouest, en d'autres termes, le trafic entre serveurs au sein d'un même datacenter.
Le modèle 3-tiers ne répond pas très efficacement à ce genre de flux simplement à cause du fait qu'une machine devrait, dans le cas du schéma ci-dessus, passer par 5 équipements pour en contacter une autre.

La solution réside donc dans la mise en place d'une architecture spine-leaf aussi connue sous le nom de distributed core.

La règle de base est que toutes les leaves sont connectées à toutes les spines, par conséquent, les spines représentent les points d'interconnexion entre les leaves. Les spines ne sont pas directement connectées entre elles et en aucun cas les leaves ne sont connectées entre elles. Voir Figure 26

[44] Littéralement « épine – feuille »
[45] En anglais, three-tier architecture
[46] Source image et aspect général : http://thenetworksurgeon.com/cisco-spine-and-leaf-architecture-discussion-nexus-5500-vs-6001/

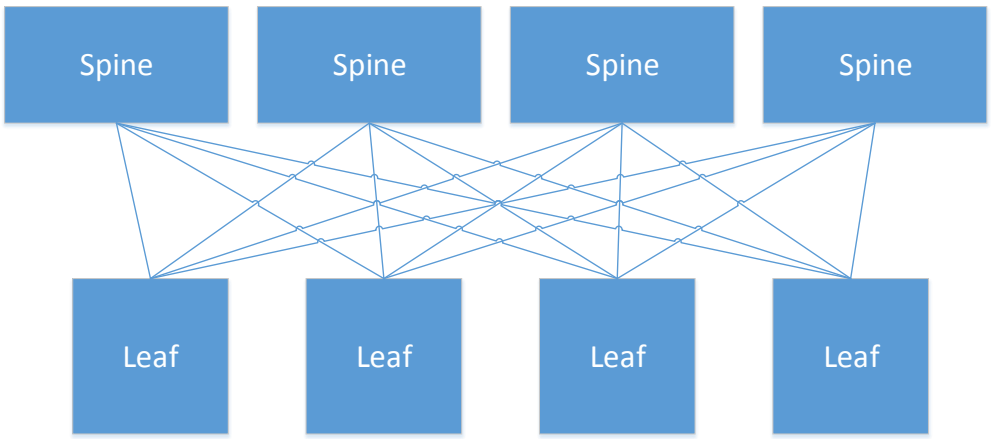


Figure 26 Exemple de Fabric ACI à 4 spines et 4 leaves

D'un point de vue réseau, on peut voir la fabric comme un switch géant.
Avec une telle infrastructure, lorsque deux serveurs communiquent ils traversent un nombre réduit et constant d'équipements, le réseau est donc déterministe : la latence est connue.

L'avantage d'une architecture « Spine – Leaf » est son niveau élevé de redondance. En cas de perte d'un lien, on ne perd qu'une partie de la bande passante pour la leaf concernée (dans l'exemple ci-dessus, un lien perdu = ¼ de la bande passante perdue pour cet équipement). A même titre, la perte d'un spine fait perdre une partie de la bande passante du backbone sans pour autant gêner d'avantage le réseau.

Pour finir, elle peut facilement être étendue soit en ajoutant une leaf si on veut augmenter le nombre d'end-points dans notre réseau soit en ajoutant un spine si on veut augmenter la capacité du backbone.[47]

2.2.2 L'APIC

Connecté aux leaves, l'APIC (Application Policy Infrastructure Controller), est un cluster d'au moins 3 équipements afin d'assurer la redondance (n+2). Comme son nom l'indique, son rôle est de contrôler l'infrastructure ACI, c'est lui qui va être le point central de configuration de la fabric, il va permettre de configurer les policies réseau que j'aborde plus loin dans ce document et il permettra d'analyser la source d'éventuels problèmes.

Son utilisation se fait via un GUI web, mais Cisco a également mis au point une API permettant, par exemple, à une entreprise d'intégrer l'APIC à son système de management actuel ou encore d'accélérer certaines tâches en créant des scripts répondant aux besoins particuliers de l'organisation.

[47] Source : https://www.youtube.com/watch?v=Z9eZYM7J33s

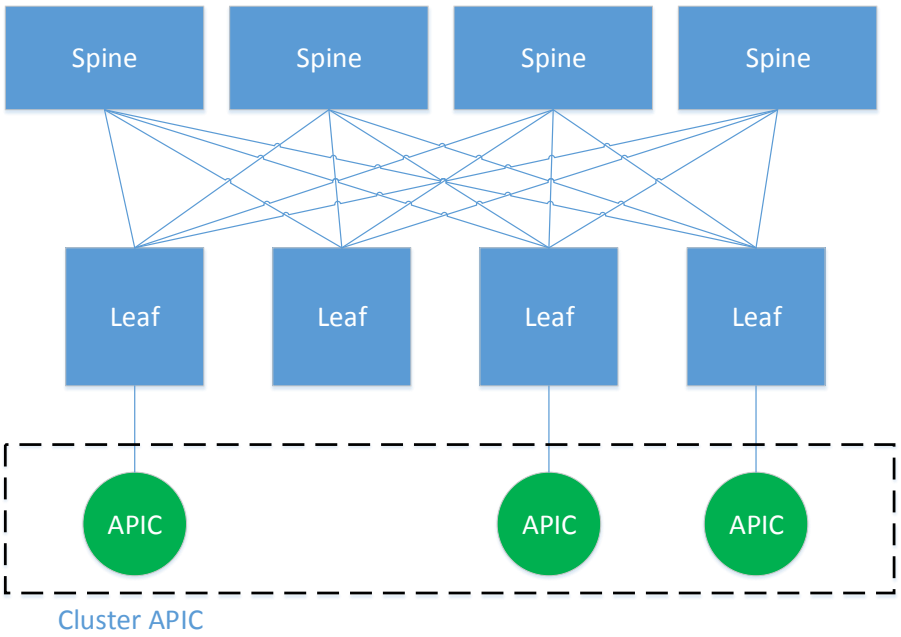


Figure 27 L'APIC est un cluster de 3 machines au moins connectées aux leaves

2.2.3 Les autres éléments réseaux

Tous les autres éléments du réseau viennent se connecter sur les leaves de la fabric quel que soit leur nature : nœuds de calcul, de stockage, équipement réseau physique (switches, routers) ou encore les liaisons vers l'extérieur, etc. Il n'est techniquement pas possible de connecter des équipements autres que des leaves aux spines. Même si cela marchait, ce serait contraire au principe de l'architecture spine-leaf.

Une des grandes forces d'ACI est, par ailleurs, la prise en charge de nombreuses solutions de virtualisation telles que VMware, Microsoft Hyper-V, Xen, OpenStack et bien d'autres. Elle va, par exemple, supporter la technologie VMotion developpée par VMware qui permet le déplacement à chaud d'une machine virtuelle sans interruption.
Le système de Cisco permet également d'intégrer les équipements qui agissent dans les couches OSI 4 à 7 comme, par exemple, des loadbalancers grâce au principe de *Service Graph* que j'aborde au chapitre suivant.

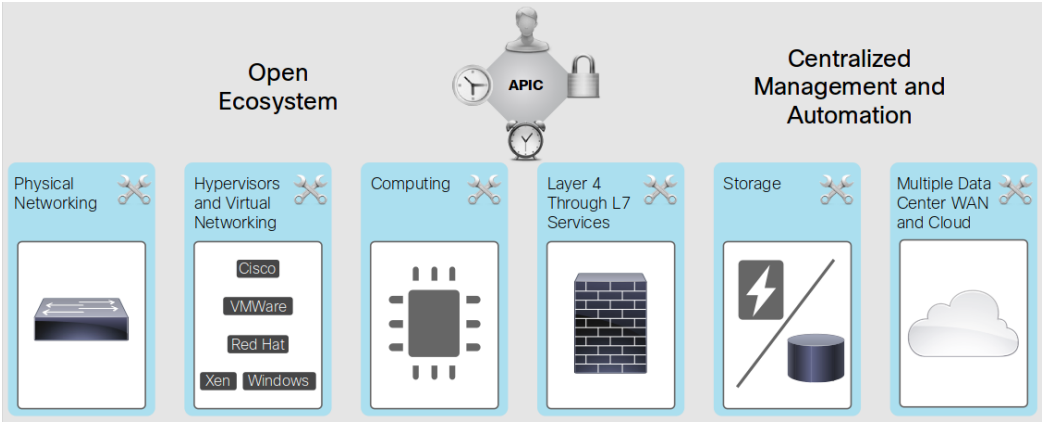


Figure 28 Cisco reconnait de nombreux équipements

2.3 Policy-model

Afin de donner toute la puissance et tout son sens à l'infrastructure ACI, Cisco se base sur un système de policies. Le principe est simple : par défaut, aucune communication n'est possible à travers la fabric ACI sans policy, il s'agit donc d'une approche à liste blanche dont les ports se comportent comme des firewalls stateless.

2.3.1 Les EPG

Afin de permettre la communication entre différents éléments du réseau, le système se base sur le concept d'End-Point Groups (EPG) qui sont des groupes logiques de nœuds (des machines physiques, virtuelles etc.). Ces groupes sont définis dans le contrôleur (l'APIC)[48].

Pour faire ce regroupement, l'administrateur dispose de différents critères : par VLAN, ports physiques, ports virtuels (remontés par des solutions supportées telles que Cisco AVS[49], VMware vSwitch ou les VxLAN), ou encore les IP.

Pour illustrer ceci, prenons un exemple classique : une application web. Ce genre d'application est généralement divisé en 3 parties : les serveurs web qui fournissent les pages, les serveurs de l'application qui exécutent des algorithmes en vue de fournir un résultat et des machines faisant tourner la base de données qui stocke les informations utiles. Dans ce cas, l'administrateur créera 3 EPG : un groupe pour les serveurs web, un pour les serveurs applicatifs et le dernier pour les serveurs de base de données.

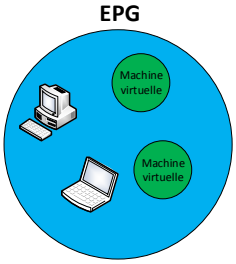


Figure 29 L'EPG regroupe différents endpoints

Une fois que les groupes ont été définis, il faut créer des contrats (Contracts).

2.3.2 Les contracts

Les contrats, dans la terminologie Cisco, sont un set de règles (policies) permettant à deux ou plusieurs groupes de nœuds (EPG) de communiquer un peu à la manière des règles que l'on configure dans un firewall, mais avec d'avantage de possibilités.

Un contrat comprend, dans les grandes lignes, des filtres (champs de couches 2 ou 4 tels que le protocole de couche 3 utilisée ou les ports TCP/UDP concernés) ainsi que des actions qui peuvent être basiques telles que « allow » ou être plus avancées comme « redirect » ou modifier le tagging QoS.

[48] Comment créer un EPG via le GUI : https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/virtualization/video/cisco_aci_scvmm_create_epg.html

[49] Cisco Application Virtual Switch : https://www.cisco.com/c/en/us/products/switches/application-virtual-switch/index.html

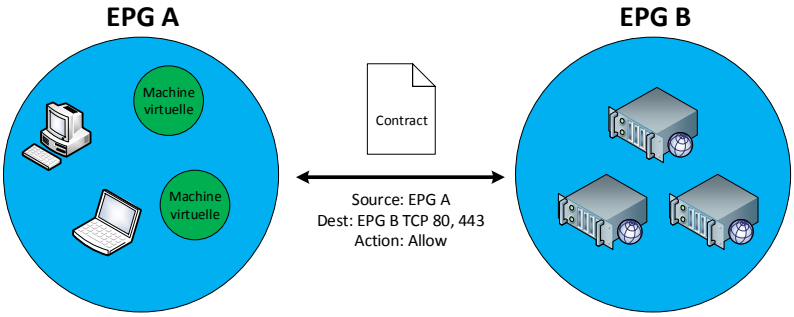


Figure 30 Exemple d'un groupe de clients accédant à des serveurs web

2.3.3 Les service graphs

Plutôt que de permettre le passage direct du trafic d'un EPG à un autre, ACI donne la possibilité de rediriger les paquets vers un équipement travaillant dans les couches 4 à 7, par exemple un firewall faisant du filtrage applicatif (niveau 7), avant de renvoyer le trafic sortant vers la destination.

La fabric ne faisant que du filtrage jusqu'en couche 4 (TCP, UDP…), les services graphs permettent d'opérer jusqu'en couche 7.

L'avantage de cette méthode est qu'elle peut permettre une certaine simplification des règles à mettre en place dans les différents équipements. Par exemple, si on veut ajouter un firewall pour bénéficier de ses fonctionnalités avancées et qu'on veut ne laisser passer que le SSH provenant de certaines adresses IP, il suffit de définir une règle pour le SSH dans le firewall. Il n'est pas nécessaire de définir les IP autorisées puisque cet aspect est géré par ACI avec les endpoint groups.

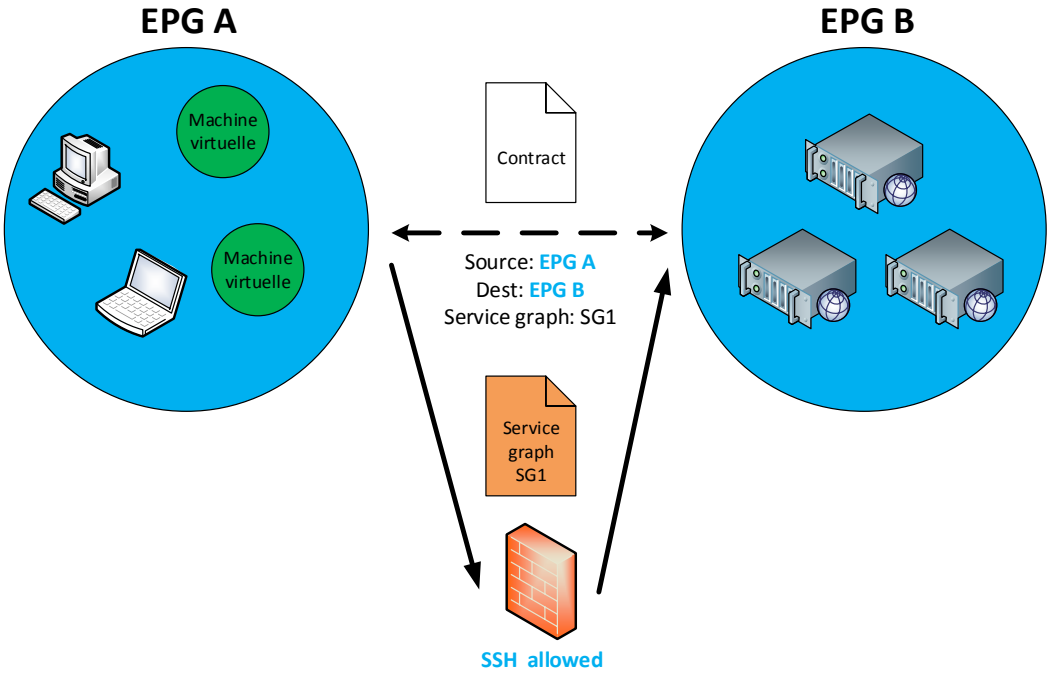


Figure 31 Exemple d'un service graph faisant filtrer le trafic à un pare-feu

2.4 Matériel

Cisco ACI nécessite des équipements particuliers de la série Nexus 9000. Il en existe principalement deux types : la série 9300 à « ports fixes »[50] et la série des 9500[5152] qui sont des châssis modulaires. Grâce à l'utilisation d'équipements spécialisés et créés pour répondre aux besoins d'ACI, Cisco assure les performances les plus optimales possibles. Une grande partie des opérations réseau sont effectuées au niveau matériel grâce à des ASIC[53] développées par la firme.

2.4.1 Nexus 9300 series

Les switches de la serie 9300 sont des équipements à « ports fixes » (c'est-à-dire que le nombre de ports est fixe à l'exception du module d'uplink qui peut être changé sur certains modèles) proposant des ports dont le débit peut aller jusqu'à 10Gbps pour certains ou 40Gbps pour les plus performants. Ils sont non-bloquants, c'est-à-dire qu'ils supportent un trafic à plein débit sur tous les ports en même temps sans occasionner de ralentissements ou de pertes.[54]

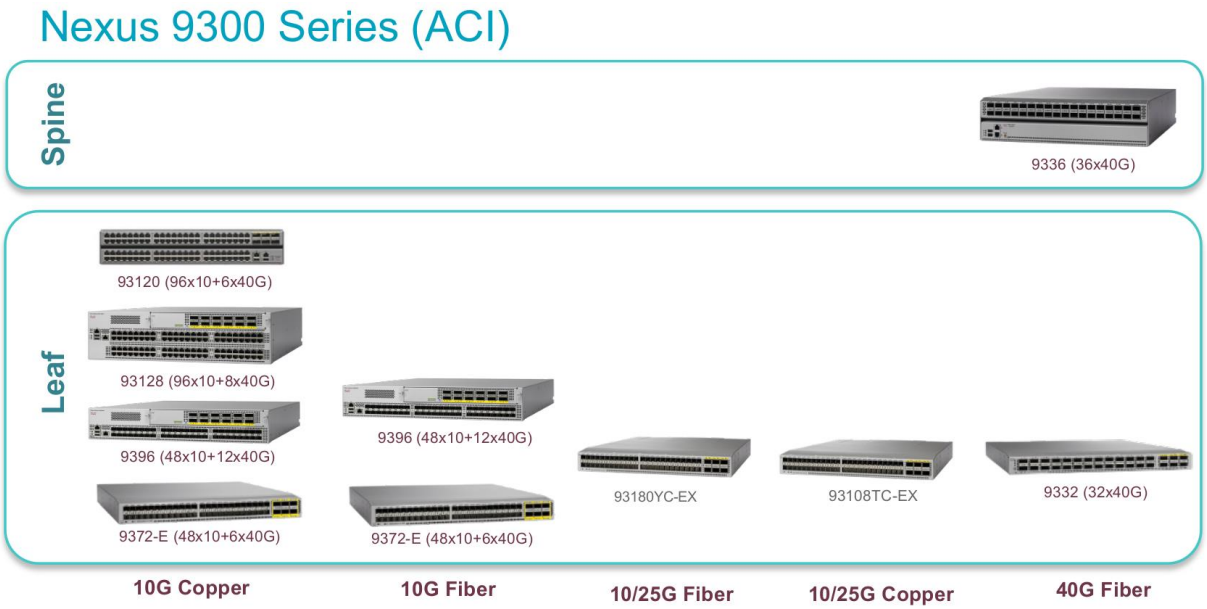


Figure 32 Les Nexus 9300 supportant ACI en 2016[55]

[50] Data sheets serie 9300: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-736967.html
[51] Data sheet n°1 serie 9500 : https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-729404.html
[52] Data sheet n°2 serie 9500: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-732088.html
[53] Application-Specific Integrated Circuit
[54] Définition de non-bloquant : http://howdoesinternetwork.com/2015/what-is-a-non-blocking-switch
[55] Source : Slides présentés par Cisco à Rolle lors du workshop Datacenter du 27 mai 2016

2.4.2 Nexus 9500 series

Pour les réseaux de plus grande envergure et afin de répondre à des besoins plus particuliers, Cisco propose les Nexus 9500 qui sont des châssis de 4, 8 ou 16 slots qui peuvent être équipés de différentes line cards de 8 à 48 ports allant de 1 à 100 Gbps.

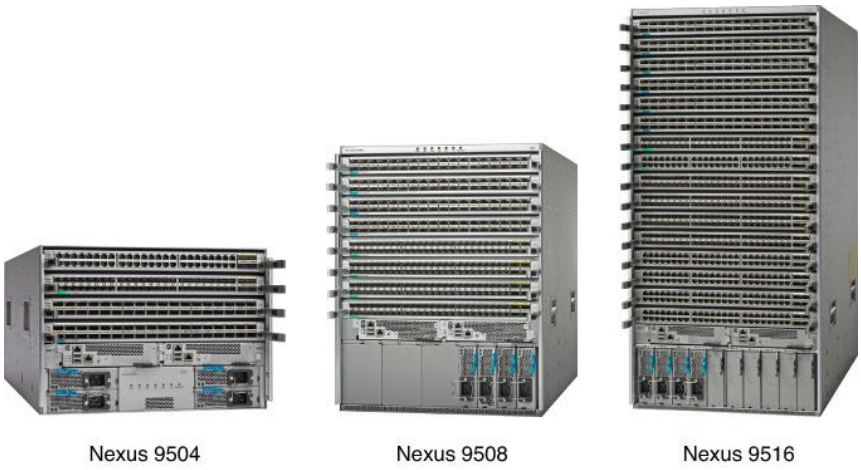


Figure 33 Les châssis de la gamme Nexus 9500[56]

Il est à noter que les châssis de 4 et 8 slots peuvent accueillir une line card de 36 ports de 40 Gbps non-bloquants[57], un des rares équipements sur le marché à atteindre de telles performances.

Figure 34 Exemple de line card (en l'occurrence la 9K-X9736PQ)[58]

Ces line cards sont interconnectées entre-elles à l'intérieur du châssis grâce à des fabric modules sur lesquelles elles se connectent directement alors qu'habituellement, ce genre de châssis est équipé d'un back-plane permettant l'interconnexion. Cette particularité permet un meilleur passage de l'air et donc un meilleur refroidissement.

[56] Source : https://www.safaribooksonline.com/library/view/ccna-data-center/9780133860429/graphics/02fig03.jpg
[57] Référence : N9K-X9536PQ : https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-736677.html
[58] Source : http://media.mwnewsroom.com/el-paso-times/cisco-delivers-on-aci-vision-adds-aci-migration-path-traditional-cisco-data-centers-1840638

Figure 35 Fabric module Nexus 9000[59]

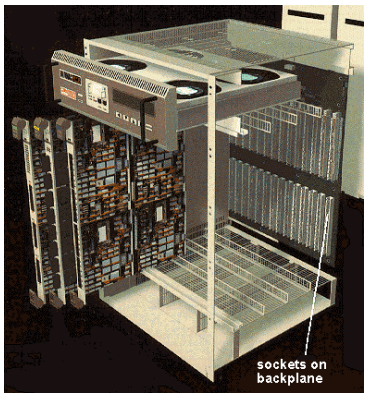


Figure 36 Backplane dans un ancien équipement de la marque Cabletron[60]

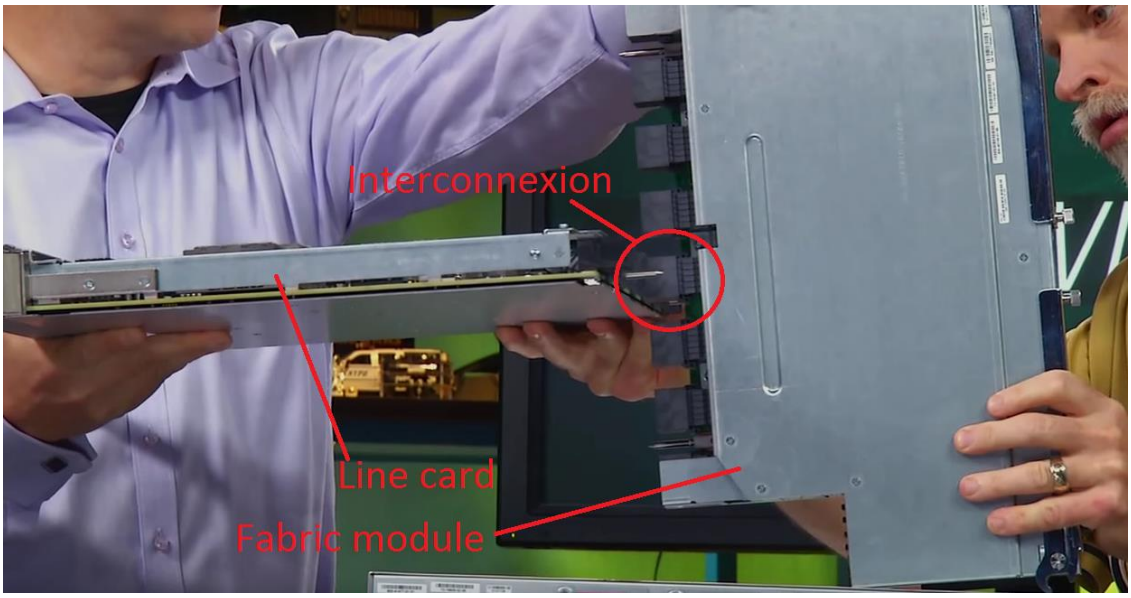


Figure 37 Présentation de l'interconnexion entre une line card et un fabric module[61]

[59] Source : https://clnv.s3.amazonaws.com/2015/usa/pdf/BRKDCT-3101.pdf
[60] Source : http://www.yourdictionary.com/modular-chassis
[61] Source : https://www.youtube.com/watch?v=youleTI9p8U

En plus des lines cards, des fabric modules, des blocs d'alimentations et des ventilateurs, les châssis contiennent 2 éléments essentiels : le contrôleur et le superviseur[62].

Le superviseur est le cerveau du châssis, il est chargé du « control-plane »[63], en d'autres termes, du gros du travail (Protocoles et tables de routage, etc.). C'est un équipement puissant (processeur multi-cœurs, mémoire importante etc.).

Figure 38 Superviseur Nexus 9500 series

Le contrôleur, quant à lui, est simplement responsable du management du châssis, c'est-à-dire qu'il en chargé de la gestion de l'alimentation, de la régulation des ventilateurs etc.

Figure 39 Contrôleur Nexus 9500 series[64]

2.4.3 L'APIC

L'Application Policy Infrastructure Controller (APIC) est le point central pour l'administration des policies, pour le monitoring et l'automatisation du provisionnement réseau.

L'APIC se présente toujours sous la forme d'un cluster de 3 machines minimum afin d'assurer la redondance en cas de problème (redondance N+2). Il faut toutefois noter que si le cluster venait à tomber, le réseau ACI (la fabric) continuerait à fonctionner, mais la mise à disposition de nouvelles applications ne serait plus possible.

D'un point de vue matériel, il s'agit d'un Cisco UCS 220 M1, M3 ou M4 proposé avec différentes configurations matérielles en fonction des besoins.[65]

[62] Bien entendu, ces modules sont redondants.
[63] Définition : https://en.wikipedia.org/wiki/Control_plane
[64] Source : https://clnv.s3.amazonaws.com/2015/usa/pdf/BRKDCT-3101.pdf
[65] Le matériel en détail : https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/hw/aci_hig/guide/b_aci_hardware_install_guide/b_aci_hardware_install_guide_chapter_01.html

Figure 40 L'Appliance APIC[66]

2.5 Gestion du trafic

2.5.1 Rôle des leaves et spines

Au sein de la fabric, les leaves et les spines ont un rôle bien défini et différent.

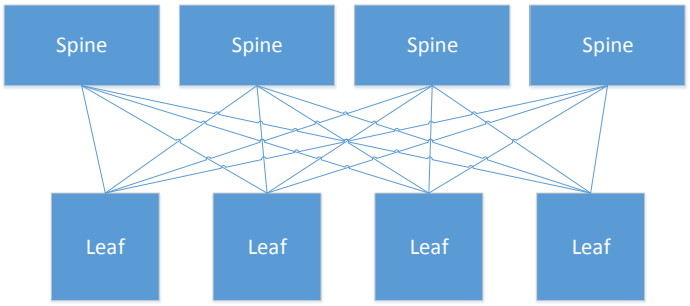


Figure 41 Rappel de l'architecture spine-leaf

Grâce à un processus de découverte des nœuds rattachés à la fabric, les spines connaissant l'ensemble de leur adresse et de leur emplacement dans le réseau. Elles possèdent une table pouvant compter jusqu'à 1 millions d'entrées d'adresses physiques (MAC) et d'IP de nœuds.

De leur côté, les leaves ne connaissent qu'une partie de ces informations, elles ont un cache local leur permettant de connaître les nœuds qui leur sont rattachés ainsi que les nœuds connectés ailleurs dans la fabric avec lesquels il y a eu un échange de trafic récemment (On parle alors de data driven learning).

On peut donc voir la fabric ACI comme un grand switch dont la table ARP n'est pas unique, mais répliquée sur différents équipements. La solution implémentée par Cisco utilise la mémoire de la leaf recevant le trafic à transférer pour renseigner, dans le paquet, l'adresse de la leaf de sortie avant de l'envoyer à une des spines pour forwarding selon un mécanisme de load balancing.

En revanche, si la leaf n'a pas d'enregistrement en mémoire lui indiquant le point de sortie auquel faire suivre le trafic, elle renseigne l'adresse d'une spine comme destination et c'est la spine qui va utiliser sa connaissance des nœuds pour renseigner l'adresse de la leaf de sortie, un peu comme le ferait un routeur lorsqu'il indique le prochain saut lorsqu'il route du trafic.

[66]Source :https://www.cisco.com/c/dam/en/us/td/docs/unified_computing/ucs/UCS_CVDs/Cisco_ACI_Architecture_for_BigData_with_Cloudera.pdf

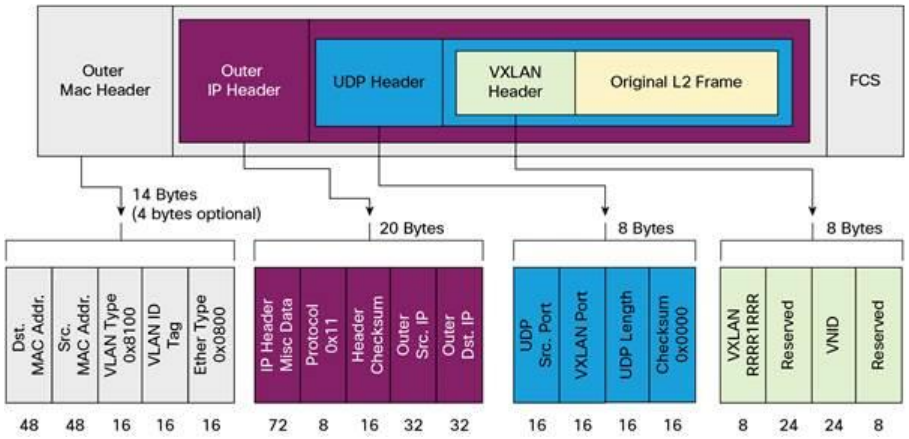
2.5.2 Les VxLAN

Cisco se base beaucoup sur les VxLAN (Virtual eXtensible Local Area Network), il est donc nécessaire d'en avoir un aperçu afin de comprendre comment ils sont utilisés par ACI.

Cette technologie présentée en 2011 lors du VMworld à Las Vegas [67] et développée par de grands groupes tels que Cisco, Vmware, Citrix ou encore RedHat est décrite par la RFC 7348[68] comme étant une structure permettant d'encapsuler des réseaux de couche 2 sur des réseaux de couche 3. Dans la pratique, nous retrouverons donc de l'ethernet encapsulé dans de l'UDP. De cette manière, la couche 2 qui est normalement confinée par les routeurs pourra être routée, ceci permet donc d'étendre un réseau de couche 2 au-delà des limites habituelles.

De plus, cette pratique permet d'isoler différents réseaux partageant le même réseau physique et/ou les mêmes adresses IP.

Comme en témoigne, le schéma d'empilement des couches suivant, l'overhead est de 50 octets, ce qui est négligeable par rapport au MTU (Maximum Transmission Unit) de 9000 bytes (jumbo frames) configuré par défaut sur les équipements ACI (environ 0.5% d'overhead).



Grâce à un VNID (Virtual Network IDentifier) d'une longueur de 24 bits, il est possible de créer plus de 16 millions de VxLAN différents, ce qui répond parfaitement aux exigences d'un cloud de grande envergure qui comprend de nombreux tenants et de nombreuses applications qui doivent être isolés. En comparaison, il est possible de créer « seulement » 4094 VLAN.

[67] Annonce officielle : https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=467114
[68] https://tools.ietf.org/html/rfc7348

2.5.3 Normalisation du trafic

En dehors de la fabric ACI, le trafic est très hétérogène : certains paquets sont, par exemple, encapsulés dans des VLAN ou des VxLAN etc. voire sans encapsulation particulière. Afin d'implémenter son système de policies, Cisco a décidé d'utiliser des VxLAN au sein de la fabric.

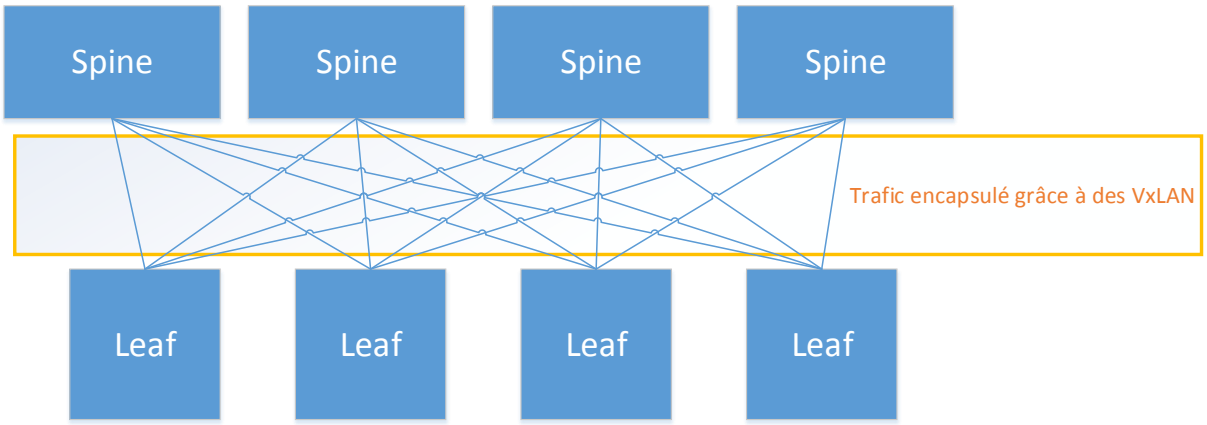


Figure 42 Utilisation de VxLAN afin d'uniformiser

Dans le but d'éviter une double encapsulation, les leaves se débarrassent de l'encapsulation qui pourrait exister à l'entrée de la fabric (par exemple d'un VLAN), le trafic est alors encapsulé dans un VxLAN à travers le réseau puis désencapsulé à la sortie afin de continuer sa route, voire ré-encapsulé dans un VLAN (ou autre tel qu'un VxLAN ou dans une trame NVGRE).
Le trafic est donc uniformisé au sein de la fabric ACI. Le but de cette opération est d'éviter une double encapsulation dans la fabric, ce qui représenterait un overhead supplémentaire qu'il est intéressant de minimiser pour optimiser les performances du système.
En outre, comme nous le verrons plus loin, les VxLAN permettent de transporter, dans leur entête, des informations concernant l'application des policies ACI.

Le schéma ci-dessous représente un exemple pratique dans lequel le serveur A envoie des données au serveur B. Les deux serveurs se trouvent dans le même VLAN (ce qui n'est pas obligatoire pour que cela fonctionne). Pour le passage dans la fabric, la leaf se débarasse du VLAN, encapsule le trafic dans un VxLAN et le trafic retrouve son VLAN à la sortie.

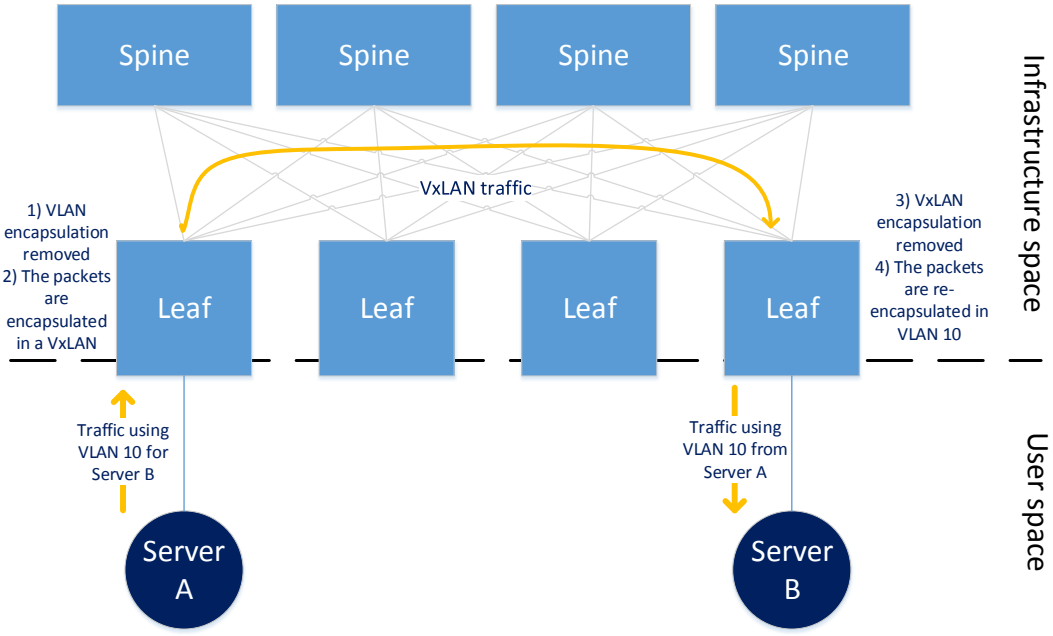


Figure 43 Exemple de l'uniformisation du trafic au sein de la fabric

2.5.4 ARP flooding

Un des protocoles bien connus dans les réseaux locaux est l'ARP (Address Resolution Protocol) qui permet à un nœud de découvrir l'adresse physique d'un autre nœud qui se trouve sur le même réseau à l'aide de son adresse IP.
Comme la machine ne connaît pas l'adresse MAC de sa destination, ce protocole fait des requêtes en broadcast, ce qui peut devenir un cauchemar pour les administrateurs réseaux travaillant sur de grands réseaux. Voyons comment ACI propose de minimiser ceci.

Dans notre cas, étant donné l'envergure de la fabric ACI, il est capital d'éviter d'avoir de l'ARP flooding dans la fabric afin de préserver la stabilité du réseau et de ne pas surcharger inutilement les équipements.
Puisque les leaves, ou du moins les spines, connaissent l'emplacement des endpoints, il ne sert à rien de flooder. Une requête ARP sera donc, comme le reste du trafic, encapsulée dans un VxLAN en entrant dans la fabric. La destination VTEP (Virtual Tunnel End Point autrement dit le point de sortie de la fabric, i.e l'adresse d'une leaf) sera renseignée soit par la leaf soit par la spine puis transmis à la leaf auquel l'équipement dont l'adresse MAC est recherchée est attaché.

La requête est, finalement, débarrassée du VxLAN qui lui a permis de traverser la fabric en unicast et elle arrive au destinataire qui répondra en unicast comme le protocole le prévoit.[69]

[69] Source : https://www.youtube.com/watch?v=wIYJMOD261k

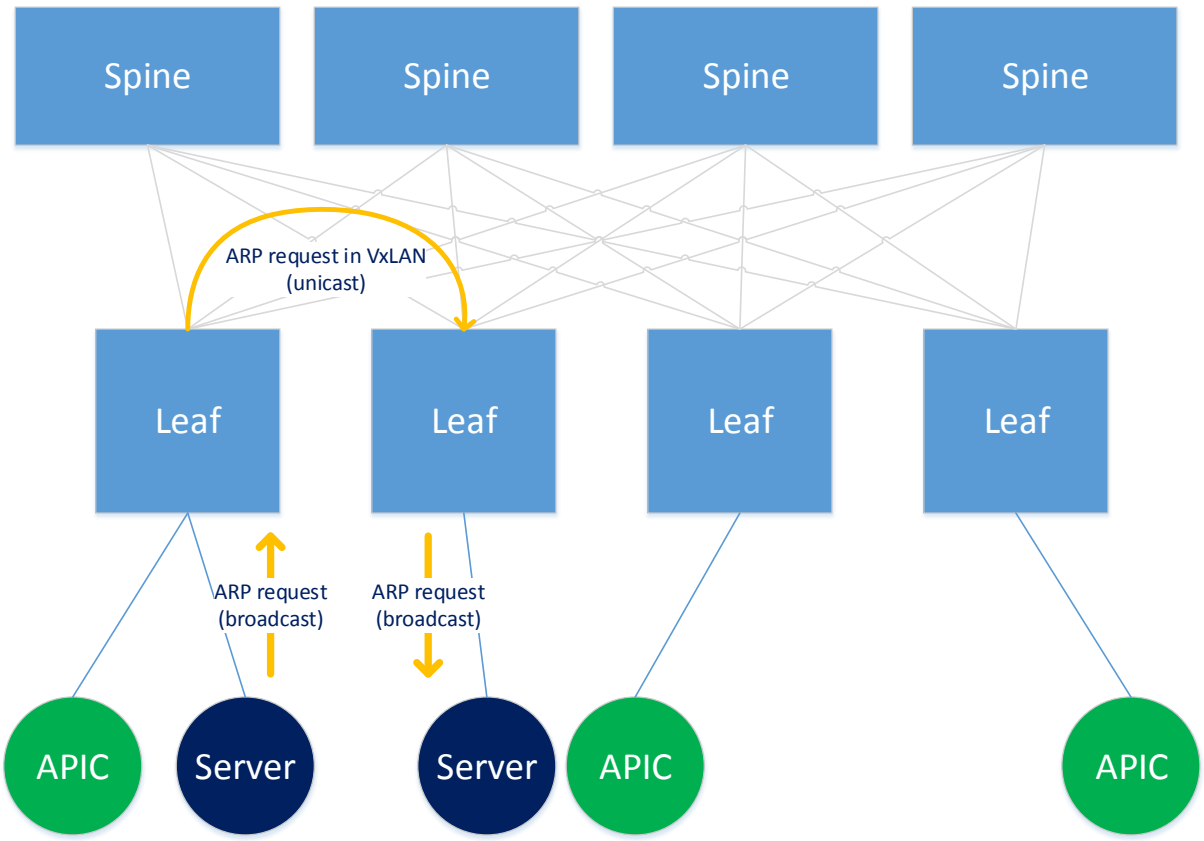


Figure 44 Le parcours d'une requête ARP à travers la fabric

2.5.5 Application des policies

Une fois que l'administrateur a configuré sur l'APIC les policies nécessaires à l'application qu'il déploie, elles sont appliquées par la fabric. Dans l'infrastructure ACI, ce sont les leaves qui sont chargées d'effectuer cette opération.
Dans l'absolu, la compréhension de l'administrateur d'un tel système peut s'arrêter là, il peut imaginer le reste comme étant une grande boîte noire qui fait ce qui a été paramétré dans l'APIC, mais il est intéressant de comprendre d'avantage ce qu'il se passe afin d'avoir une idée du travail réalisé par Cisco et afin de pouvoir analyser ce qu'il se passe en cas d'apparition de problèmes.

Les leaves font du data driven learning[70] afin de découvrir les nœuds qui leur sont rattachées et obtiennent les policies correspondantes auprès du contrôleur (APIC). De cette manière, chaque leaf est informée des règles concernant les end-points en dessous d'elle.

Lorsque du trafic à destination d'une autre machine se trouvant sur une autre leaf lui parvient, la leaf encapsule le paquet dans un VxLAN comme décrit précédemment et elle utilise les différents champs de ce protocole afin de transférer le paquet et de lui appliquer les règles (policies) qui le concerne.[71]
Si elle connaît l'EPG de destination, elle est capable d'appliquer directement les policies ad-hoc, elle passera alors un bit (Policy Applied bit) de l'entête VxLAN à 1. Dans le cas, contraire, elle transmettra le paquet en laissant le bit à 0.

[70] Apprentissage en analysant le trafic entrant
[71] Source : https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731310.html

Etant donné que le VxLAN est un protocole faisant de l'encapsulation, l'entrée et la sortie du tunnel doivent être spécifiés afin que le paquet soit transporté, encapsulé et désencapsulé correctement. Pour ce faire, on utilise l'en-tête du protocole IP et plus particulièrement les champs « Source IP » et « Destination IP » (aussi appelés « Source VTEP » et « Destination VTEP »[72], points [a], [b], [c] et [e] de la Figure 46).

De plus, afin de spécifier le « Policy Applied bit » (voir Figure 46 points [a], [c] et [d]) et de spécifier l'identifiant permettant de connaître les policies à appliquer (src EPG), Cisco a soumis, en avril 2016, un draft[73] à l'IETF. Il décrit une variante des 2 premiers champs de l'entête VxLAN pour satisfaire les besoins d'ACI.

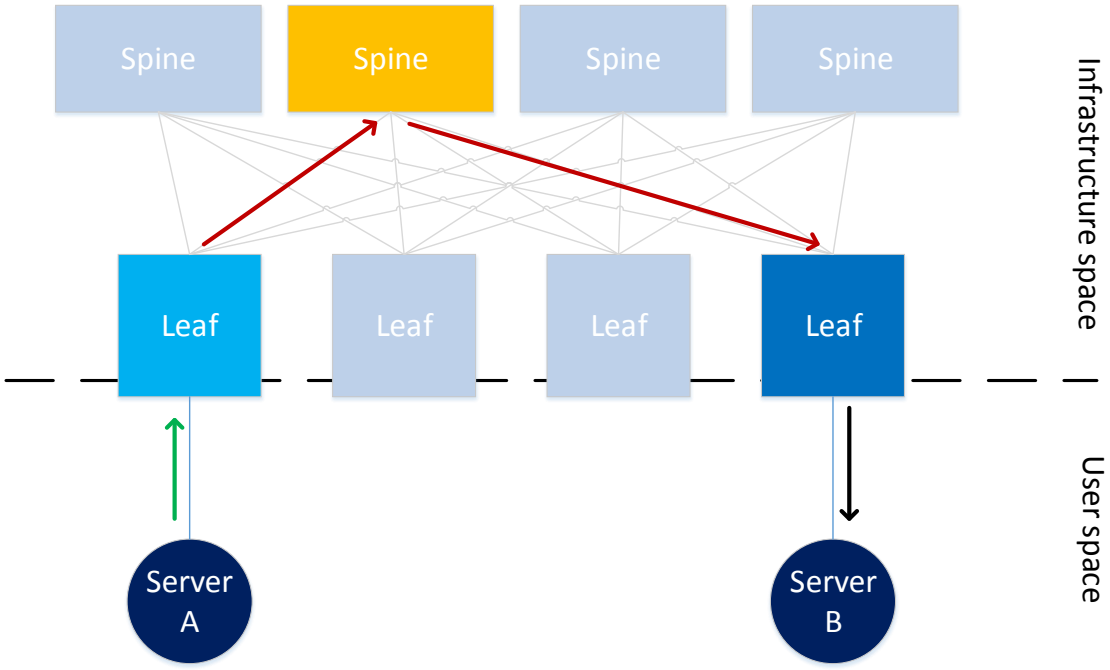


Figure 45 Schéma d'un flux unicast à travers la fabric ACI

[72] VTEP = Virtual Tunnel End-Point
[73] Draft VxLAN-GBP : https://www.ietf.org/id/draft-smith-vxlan-group-policy-02.txt

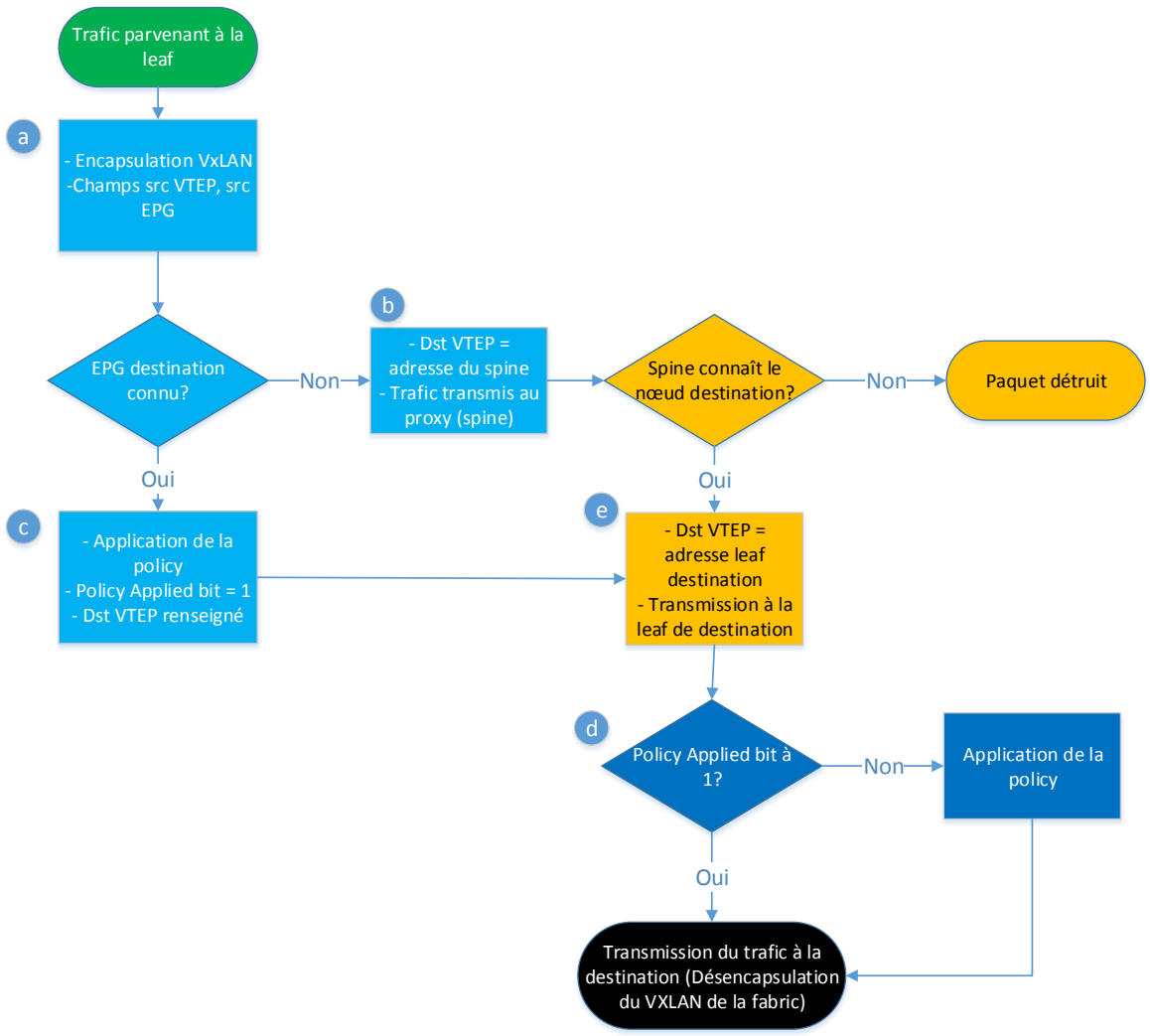


Figure 46 Flowchart de l'application des policies (voir couleurs schéma précédent)

2.6 Conclusion

Avec ACI, Cisco entend répondre aux besoins principaux du marché tout en ouvrant de nouvelles perspectives.

Le modèle implémenté est, en premier lieu, un avantage en termes de sécurité grâce à son approche par liste blanche, la visibilité qu'il apporte en un point centralisé qu'est l'APIC ainsi que la simplicité d'audit qui en découle.
Cette manière d'aborder le monde du datacenter apporte également un certaine facilité à faire du troubleshooting grâce à son interface qui permet également de gérer la majorité du réseau à partir d'un même endroit.

Un autre point fort de ce système est l'intégration facilitée grâce à l'API[74] proposée et grâce au support des équipements/solutions issues d'autres firmes (F5, VMware, Microsoft etc.), c'est

[74] Documentation de l'API REST : https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/api/rest/b_APIC_RESTful_API_User_Guide.pdf

d'ailleurs la particularité d'ACI, alors que les autres firmes telles que VMware ne se concentrent que sur l'interconnexion de leurs propres produits.

En outre, il ne faut pas négliger les efforts que la multinationale a mis dans le matériel qui a été créé spécifiquement dans le but de répondre à des besoins spécifiques avec les meilleures performances possibles. Il permet d'ouvrir de nouveaux horizons avec des projets tels que Cisco Tetration Analytics[75] qui se base sur des capteurs propres aux Nexus 9000 (comportant le suffixe -X ou -EX) pour faire de la télémétrie avancée offrant des informations très riches concernant la nature du trafic permettant même de capturer des mois de trafic (entêtes des trames) afin de faire de l'analyse à long terme.

D'un point de vue purement financier, il est, par ailleurs, intéressant de souligner que les entreprises qui ont implémenté ACI ont, selon les chiffres diffusés par Cisco, eu un important ROI (Return On Investment) notamment grâce à un cycle de développement accéléré, à des réseaux plus efficaces et à une réduction des frais de gestion du réseau désormais plus centralisé.

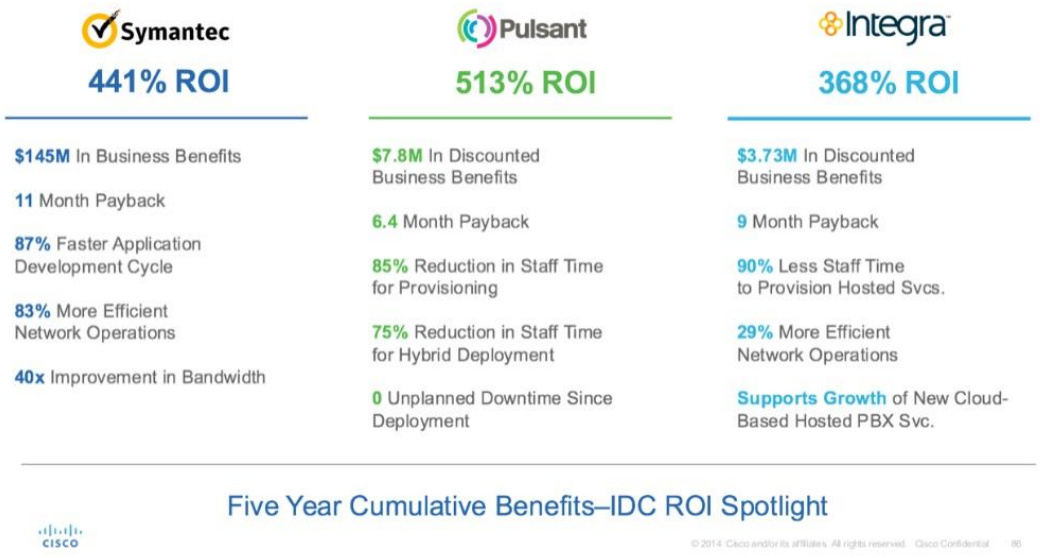


Figure 47 Retour sur investissement de l'implémentation d'ACI

[75] https://www.cisco.com/c/en/us/products/data-center-analytics/tetration-analytics

En revanche, comme ce document le montre, la terminologie et les spécificités propres à ACI ne manquent pas, pour maîtriser un tel système, il est, par conséquent, nécessaire de l'avoir bien étudié et probablement de passer par Cisco afin de suivre une formation avancée pour tirer parti de sa pleine puissance.

Finalement, l'arrivée de telles technologies rappelle qu'il est important, dans le monde des technologies, de savoir se remettre en question et de suivre les avancées technologiques. Cette solution pourrait bien annoncer un changement radical dans le monde du réseau, un professionnel qui refuserait de s'y intéresser risquerait d'être dépassé.

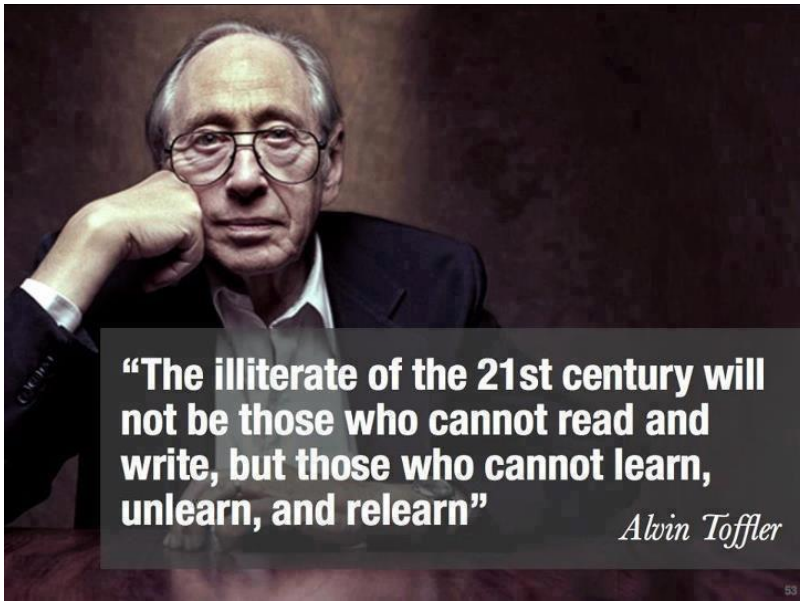


Figure 48 Citation d'Alvin Toffler[76]

2.6.1 Liens utiles

Les principaux documents officiels sont disponibles sur le portail d'ACI : http://cisco.com/go/aci

Pour aborder l'essentiel des sujets concernant ACI, Cisco met à disposition un document abordant tous les fondamentaux :
https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/aci-fundamentals/b_ACI-Fundamentals.pdf

Documentation officielle détaillée sur les questions de design :
https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731960.html

Une série de vidéo de formation a été mise à disposition par Cisco sur son site :
https://learningnetwork.cisco.com/community/learning_center/aci-training-videos/videos

[76] Source de l'image : https://gnuatheism.blogspot.com/2012/06/alvin-toffler-on-new-illiterate.html

3 Docker avec Cisco ACI

3.1 Infrastructure à disposition

3.1.1 Fabric & compute

Afin de pouvoir mettre en œuvre les différents éléments étudiés dans ce travail, Cisco m'a mis une petite infrastructure à disposition dans leurs locaux de Zürich.

La fabric est composée d'une spine (Nexus 9336PQ[77]) de 36 ports 40Gbps (QSFP+[78]) et deux leaves (Nexus 9396PX[79]) équipées chacune d'un uplink module de 12 ports de 40GbE pour la connectivité avec la spine et 48 ports de 10Gbps ethernet (SFP+[80]) pour y connecter les nœuds du réseau.
Le contrôleur, quant à lui, est un APIC M1[81].

Du côté des machines, la petite infrastructure comprend 2 serveurs UCS C220 M3 rackserver[82] équipés chacun de 2 processeurs Intel Xeon E5-2643 3.3GHz (4 cores, 8 threads) et 65 Go de RAM (8 barrettes de 8192MB).
Elles ont toutes les deux 4 disques dur de 285 Go.

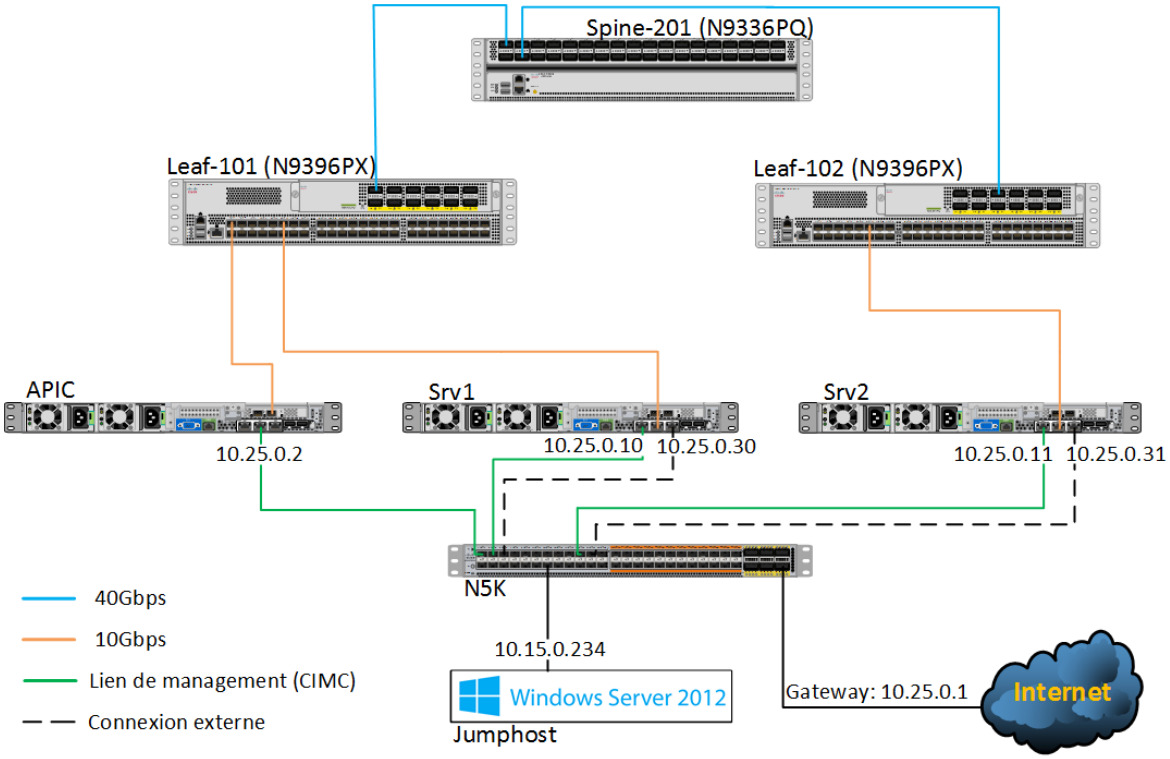


Figure 49 Schéma de l'infrastructure mise à disposition après configuration

[77] Modèle 3D du 9336PQ : https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/n9336pq-3d-model.html
[78] Modules pour la fibre supportant jusqu'à 4 canaux de 10Gbps chacun : https://en.wikipedia.org/wiki/QSFP
[79] Modèle 3D du 9396PX : https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/nexus-9396px-3d-model.html
[80] Modules pour la fibre supportant jusqu'à 10Gbps ethernet : https://fr.wikipedia.org/wiki/Small_form-factor_pluggable
[81] https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/application-policy-infrastructure-controller-apic/datasheet-c78-732414.html
[82] Modèle 3D du C220 M3 : https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/ucs-c220-m3-rack-server/ucs_c220_kaon_model_preso.html

Les deux serveurs ont accès à internet grâce à une interface directement reliée au N5K. Cette connexion a pour but de permettre l'installation des différents paquets nécessaires, le choix de cette solution permet d'éviter de configurer la fabric pour pouvoir installer les machines, ceci simplifie donc la mise en place.
Par la suite, elle va servir au management du cluster qui sera mis en place.

3.1.2 Connexion à distance

Afin d'administrer l'infrastructure à distance, l'APIC et les 2 serveurs ont, en plus d'être connectés aux leaves, l'interface de management reliée (en vert sur le schéma) à un équipement de la gamme Nexus 5000 de Cisco.

Un serveur Windows 2012 R2 est également connecté au Nexus 5k, c'est en faisant du RDP sur cette machine que j'ai eu accès au réseau mis à disposition.

La connexion RDP en elle-même est un peu particulière. En effet, elle utilise le principe de Remote Desktop Gateway qui est apparu avec Windows Server 2008[83].
Plutôt que de faire du RDP simple du client jusqu'au serveur, la connexion RDP va être transportée sur du HTTPS jusqu'à une gateway à partir de laquelle la couche SSL est mise de côté. En résumé, ce système agit un peu comme un VPN permettant ainsi, du point de vue de l'utilisateur, de passer facilement les firewalls (utilisation du port 443).
De cette manière, la connexion est protégée par une couche supplémentaire (SSL) et la gateway est authentifiée (Certificat).

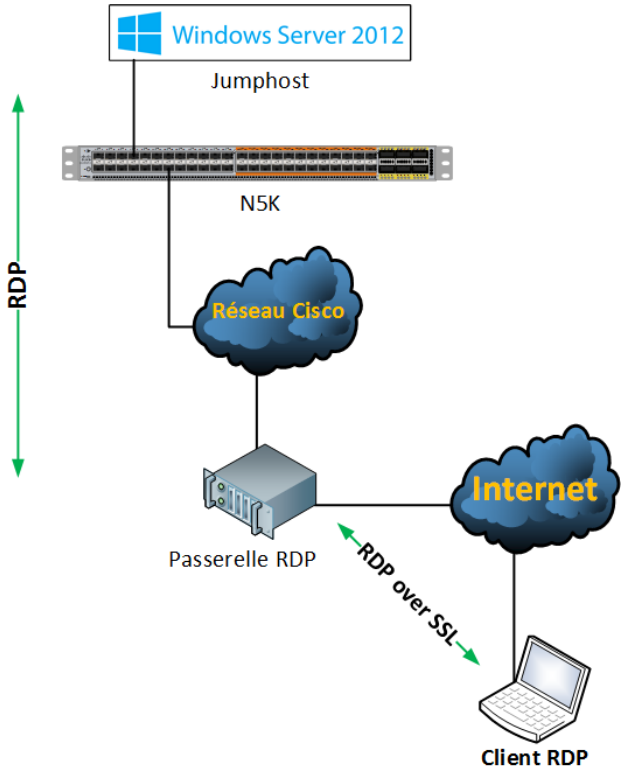


Figure 50 Connexion à distance pour accéder au labo mis en place par Cisco

[83] Technet sur Remote Desktop gateway https://technet.microsoft.com/en-us/library/dd560672(v=ws.10).aspx

3.2 Installation des serveurs

3.2.1 Configuration des disques

Les serveurs à disposition sont équipés de 4 disques dur. Par défaut, lors de l'installation d'un OS, il est impossible de les détecter. En faisant quelques recherches dans le GUI de management, le CIMC (Cisco Integrated Management Controller), j'ai découvert qu'il fallait les paramétrer.

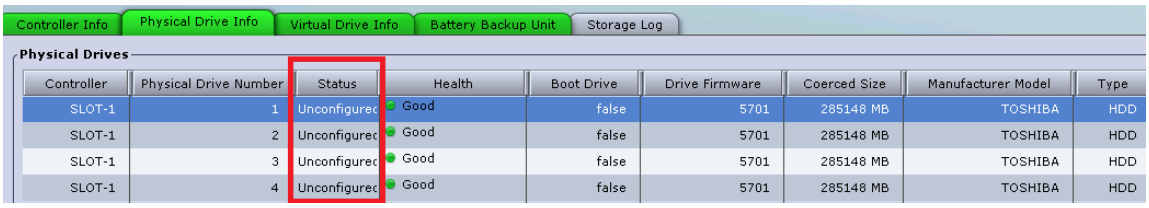


Figure 51 Les disques ne sont pas encore configurés

Il faut créer un « disque virtuel » à partir des disques physiques.

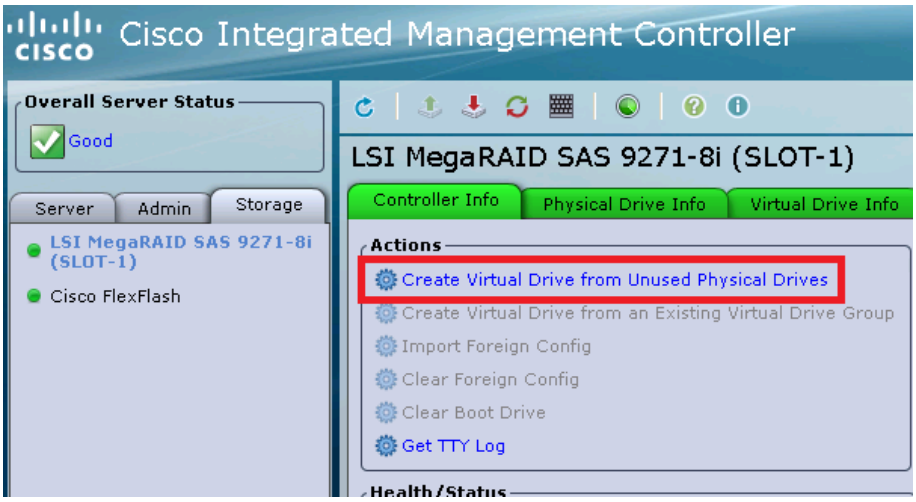


Figure 52 Le CIMC permet de créer des disques virtuels

L'interface permet alors de choisir entre différents types de RAID (0, 1, 5, 6, 10, 50, 60), il faut donc faire un choix puis sélectionner les disques physiques qui vont participer au disque virtuel.

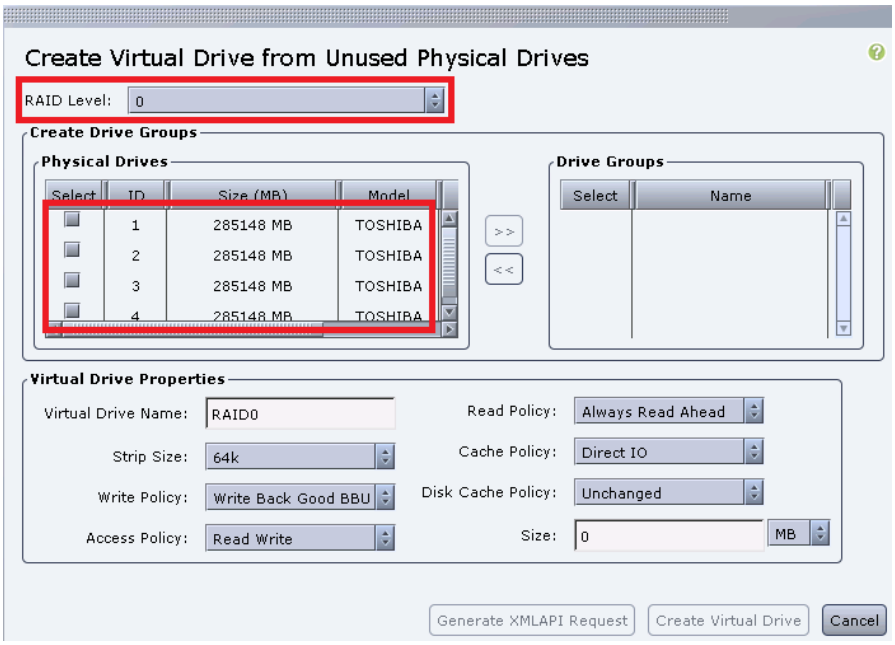


Figure 53 Création d'un disque virtuel

Une fois ces paramètres choisis, on voit que le CIMC a créé un groupe de disques avec les disques 1, 2, 3 et 4. Il en a alors créé un disque virtuel dont le nom est modifiable à même titre que le taille et différents paramètres plus spécifiques

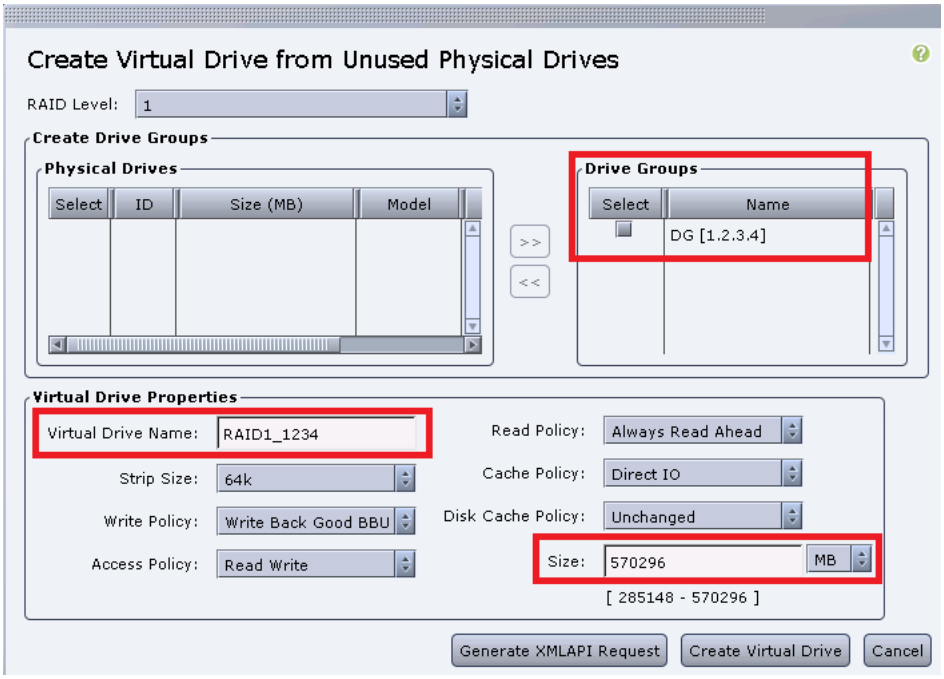


Figure 54 Les paramètres sont entrés il ne reste plus qu'à valider

En retournant dans les onglets de stockage du CIMC, on aperçoit désormais que le status des disques n'est plus « Unconfigured » mais est devenu « Online » et on peut voir que le disque virtuel a bien été créé.

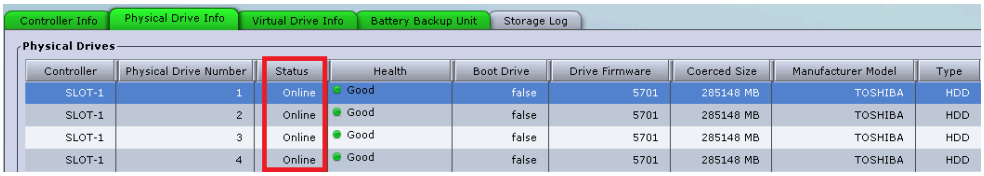


Figure 55 L'état des disques est passé à Online

Figure 56 Le disque virtuel a bien été créé

3.2.2 Installation de l'OS

La prochaine subtilité réside dans l'installation même du système d'exploitation. Il s'agit de résoudre le problème de l'installation à distance en sachant qu'aucun OS n'est installé sur la machine.

Lorsqu'un accès physique est possible, il suffit d'insérer un CD/DVD ou une clé USB préparé avec l'iso d'un système d'exploitation compatible. Dans ce cas, il a fallu trouver une autre solution.

L'interface de management (CIMC) apporte, là également, la réponse. En effet, il permet l'utilisation d'un KVM (Keyboard Video Mouse) virtuel[84]. En cliquant sur l'îcône KVM dans le CIMC, il est possible de télécharger un exécutable Java donnant ainsi la possibilité de voir ce qui s'afficherait à l'écran de la machine et d'intéragir avec celle-ci.

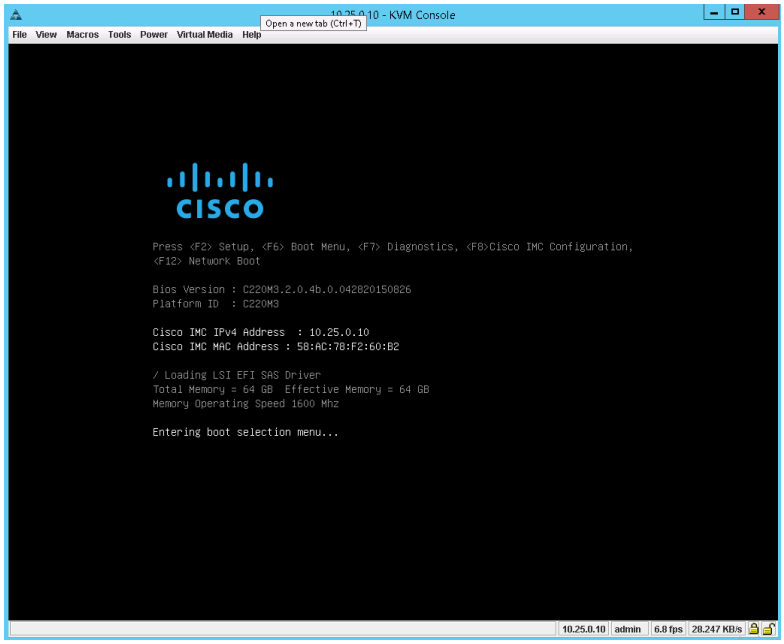


Figure 57 La console KVM proposée par le CIMC

[84] Définition de KVM virtuel : http://okvm.sourceforge.net/virtualkvm.html

Cette petite interface donne alors la possibilité de monter un disque virtuel depuis la machine avec laquelle on a ouvert le KVM.
A l'aide de Virtual CloneDrive[85], j'ai monté une ISO de CentOS 7.2.1511 (64 bits) sur le jumphost (machine à laquelle je me connecte en RDP et qui me donne accès au lab). J'ai alors pu monter le disque virtuel sur les serveurs via KVM afin d'installer le système d'exploitation.[86]

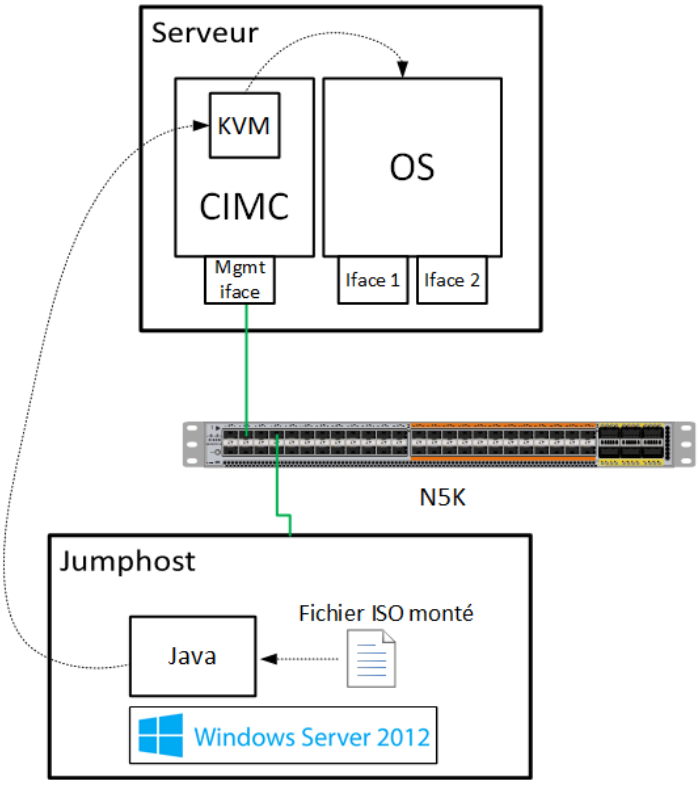


Figure 58 Schéma résumant la démarche pour installer Fedora sur les serveurs

3.3 Configuration réseau

Machine/Router	Iface management	Iface 1 (Côté fabric)	Iface 2 (Côté gateway)
APIC	10.25.0.2	-	N/A
Srv1	10.25.0.10	- (enp1s0f0)	10.25.0.30 (enp1s0f1)
Srv2	10.25.0.11	- (enp1s0f0)	10.25.0.31(enp1s0f1)

Préfixe réseau : /24
Passerelle réseau : 10.25.0.1
Adresse jumphost : 10.15.0.234

[85] Lien de téléchargement de Virtual CloneDrive : https://www.elby.ch/products/vcd.html
[86] Voir procédure «Installation à distance avec virtual KVM» en annexe (Chapitre 5.2, page 67)

3.4 Contiv

Une fois les machines installées avec leur OS, la question du lien entre Docker et ACI se pose. En effet, il faut trouver un moyen pour lier des containers à des EPG et leur appliquer des policies sans avoir à créer manuellement un lien entre ACI et Docker à chaque fois.

La solution qui supporte ceci s'appelle Contiv[87]. Contiv est un projet open-source lancé par Cisco permettant de définir des policies pour des applications basées sur des containers afin de définir le comportement de ces derniers (déclarations réseaux etc.). Le but fondamental de ce projet est d'automatiser au maximum le déploiement de containers.
Il donne la possibilité de se connecter à un contrôleur ACI (APIC) afin de le configurer et de lui permettre d'assigner les containers à différents EPG et bien sûr d'appliquer les contrats qui leur sont liés.

Bien qu'essentiel pour une collaboration entre Docker et ACI, il n'est pas qu'utilisé dans un contexte Cisco. Il peut simplement être employé sur différents ordonnanceurs de containers tels que Docker Swarm, Kubernetes, Mesos ou encore Nomad indépendamment du matériel réseau utilisé.
Il peut par ailleurs interagir avec des réseaux de couche 2, de couche 3 ou des réseaux de type overlay (e.g VxLAN).

Créé en août 2015, Contiv est principalement maintenu par une équipe de 7 personnes.

3.4.1 Installation de Contiv

L'installation est assez simple, car elle est faite à l'aide d'un script mais ne se déroule apparemment jamais sans problèmes. Il va permettre de créer un cluster Docker Swarm avec toutes les dépendances et les plugins nécessaires pour mettre en place une infrastructure de policies.

3.4.1.1 Prérequis

Cette installation requiert CentOS 7 ou Ubuntu 15 (64 bits), ce qui explique le choix de la distribution que j'ai dû faire. Ce prérequis est très limitant puisque les utilisateurs de cette solution pourraient avoir besoin d'une autre distribution pour diverses raisons. Il est explicable par le fait que Contiv est un projet très jeune.
Les machines doivent disposer de Python.

Il faut savoir que les machines prenant part au cluster que nous allons créer doivent avoir une interface reliée à la fabric ACI et une autre interface reliée à un réseau de management avec accès à internet. Seule l'interface sur ce réseau doit avoir une adresse IP, l'interface reliée à la fabric n'a besoin d'aucune configuration IP particulière.
Le but est que Contiv ait une interface pour la gestion du cluster, la création des réseaux Docker, etc. et une interface consacrée aux communications entre les containers.

[87] Site officiel de Contiv : https://contiv.github.io/

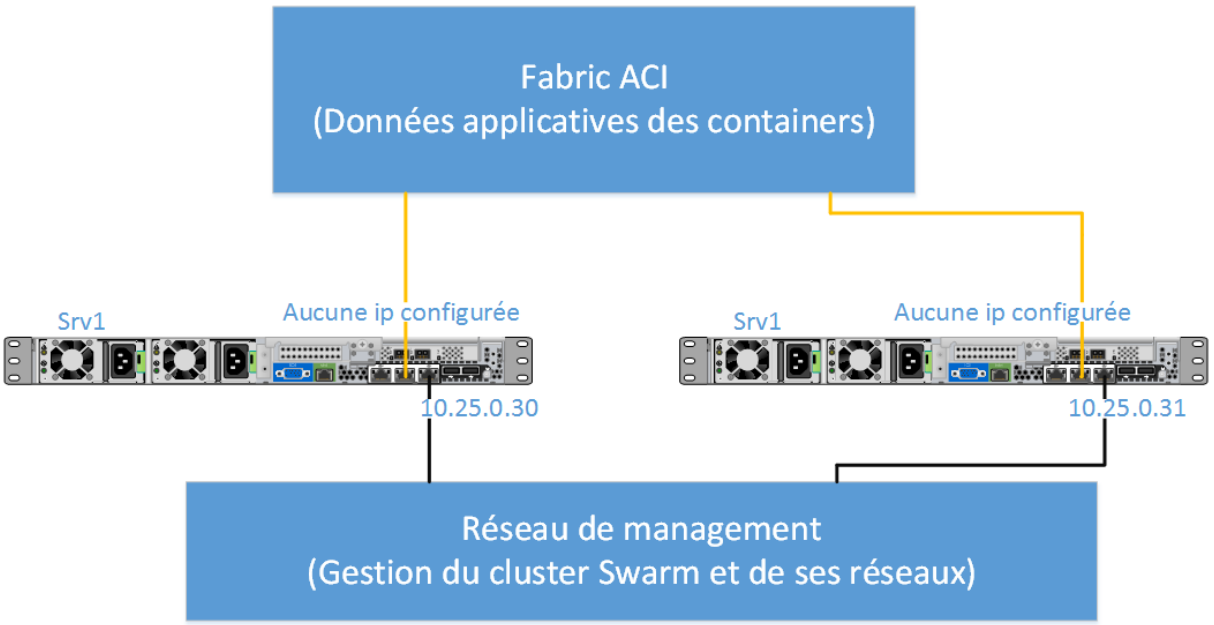


Figure 59 Configuration IP mise en place

Toutes les opérations qui suivent ne sont exécutées que sur une machine, le script se charge de faire le nécessaire sur les autres machines en SSH.

3.4.1.2 Variables d'environnement

Dans un premier temps, il faut déclarer des variables d'environnement contenant notamment les adresses des machines hôtes :

```
export CLUSTER_NODE_IPS=10.25.0.30,10.25.0.31
export no_proxy=10.25.0.30,10.25.0.31,127.0.0.1,localhost,netmaster
```

Afin de rendre l'exécution du script plus agréable, l'équipe de Contiv recommande de configurer une authentification SSH par clé entre la machine sur laquelle il est lancé et les autres machines participant au cluster (dans mon cas, il n'y a que deux machines.) ainsi que l'utilisation de sudo sans mot de passe. Sans cela, il faut entrer manuellement le mot de passe à plusieurs reprises ce qui devient vite pénible quand il faut relancer le script suite à un problème.

Etant donné que j'utilise CentOS, je n'ai pas besoin d'activer sudo sans mot de passe, car les opérations seront effectuées avec l'utilisateur root[88], en revanche, je dois configurer l'authentification SSH par clé.

[88] Alors que ce serait obligatoire avec Ubuntu puisque cette distribution ne permet pas, par défaut, le login en tant que root.

3.4.1.3 Authentification SSH par clé

Dans un premier temps, se connecter à la machine qui exécutera le script. L'utilisateur qui pourra s'authentifier sera root, il faut donc se connecter en tant que root.

Puis créer une paire de clés à l'aide de la commande suivante :

```
ssh-keygen -t rsa
```

Ensuite, créer un dossier .ssh sur le serveur distant avec l'utilisateur cible (en l'occurrence également root), exécuter cette action depuis le serveur client en ligne de commande nous fait gagner du temps :

```
ssh root@10.25.0.31 mkdir -p .ssh
```

Copier la clé publique généré dans la machine distante :

```
cat .ssh/id_rsa.pub | ssh root@10.25.0.31 'cat >> .ssh/authorized_keys'
```

Pour assurer que les différentes versions de SSH ne posent pas de problèmes, il faut encore configurer les permissions suivantes :

```
ssh root@10.25.0.31 "chmod 700 .ssh; chmod 640 .ssh/authorized_keys"
```

Il ne reste plus qu'à tester :

```
[root@srv1 ~]# ssh root@10.25.0.31
Last login: Tue Jul 26 11:48:02 2016 from 10.15.0.234
[root@srv2 ~]#
```

Figure 60 La connexion SSH ne requiert pas de mot de passe, l'authentification est faite par clé

3.4.1.4 Le script de Contiv

Le rôle du script d'installation de Contiv est de préparer les serveurs afin de créer un cluster Swarm. De manière plus précise, il se charge de configurer les règles iptables nécessaires, d'installer les packages requis (notamment Docker et Swarm), d'installer le plugin réseau Contiv pour Docker nommé à l'heure actuelle *netplugin*[89], de lancer les containers utiles au cluster et à Contiv et de faire les diverses autres configurations nécessaires.

Pour l'obtenir, un simple wget suffit :

```
wget
https://raw.githubusercontent.com/contiv/demo/master/net/net_demo_installer
```

[89] Ce nom étant mal choisi les développeurs ont annoncé qu'il sera changé en *contiv* à l'avenir

Dans le dossier courant, il faut créer un fichier de configuration contenant les informations de l'infrastructure actuelle nommé *cfg.yml*. Un modèle de configuration pour ACI est disponible sur Github[90]. Voici la configuration que j'ai créé pour mon labo :

```
## Connection Info
CONNECTION_INFO:
      10.25.0.30:
        control: enp1s0f1
        data: enp1s0f0
      10.25.0.31:
        control: enp1s0f1
        data: enp1s0f0
##================================================================
## APIC Access Info
APIC_URL: "https://10.25.0.2:443"
APIC_USERNAME: "admin"
APIC_PASSWORD: "C1sco123"
APIC_PHYS_DOMAIN: "contiv-domain"
APIC_EPG_BRIDGE_DOMAIN: "not_specified"
APIC_CONTRACTS_UNRESTRICTED_MODE: "no"

APIC_LEAF_NODES:
    - topology/pod-1/node-101
    - topology/pod-1/node-102
##================================================================
```

Quelques commentaires à propos de ce fichier :

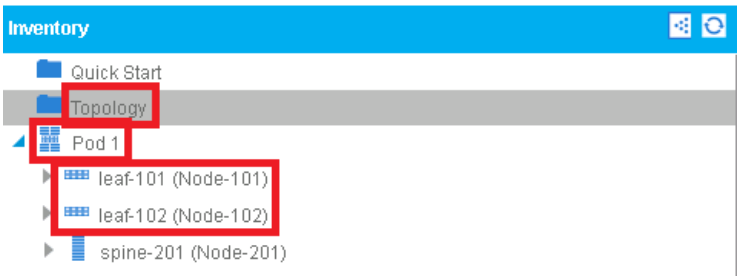
- *CONNECTION_INFO :* contient les informations de chaque hôte du cluster, l'adresse IP indiquée est évidemment celle de l'interface connectée au réseau de management (puisqu'il ne devrait pas y en avoir d'autres configurée comme indiqué plus haut).
    - *control :* correspond au nom de l'interface reliée au réseau de management
    - *data :* correspond au nom de l'interface reliée à la fabric ACI

- *APIC_PHYS_DOMAIN :* correspond au nom d'un « domaine physique » configuré dans l'APIC selon la procédure brièvement décrite sur le site de Contiv[91] (Le pool de VLAN choisi sera arbitrairement en 100 et 499).

- *APIC_CONTRACTS_UNRESTRICTED_MODE* et *APIC_EPG_BRIDGE_DOMAIN* servent respectivement à forcer l'application des policies entre EPG et à utiliser un bridge domain particulier (je ne rentrerai pas dans le détail pour ces paramètres puisqu'ils ne représentent aucun intérêt dans ce cas).

[90] https://github.com/contiv/demo/blob/master/net/extras/sample_aci_cfg.yml
[91] https://contiv.github.io/documents/gettingStarted/networking/aci.html

- *APIC_LEAF_NODES :* correspond au chemin des leaves connectées à mes serveurs, cette information peut être lue dans l'APIC tout en sachant sur quel leaves sont connectées les machines :



Afin de disposer de la dernière version du plugin réseau, il faut impérativement ajouter une variable d'environnement contenant la référence de celle-ci[92] :

```
export contiv_network_version="v0.1-06-17-2016.08-42-14.UTC"
```

Sans cette variable, Contiv ne pourra pas interagir avec l'APIC, l'oubli de cette étape m'a fait perdre beaucoup de temps, car aucune erreur explicite n'apparaît.

Finalement, il ne reste plus qu'à donner les droits nécessaire au script et à le lancer pour que le nécessaire soit mis en place. L'option *-a* indique que le mode à utiliser est ACI :

```
chmod +x net_demo_installer

net_demo_installer -a
```

A ce niveau, ni Rolf Scharer de Cisco à Zürich ni moi-même n'avons réussi à faire fonctionner ce script du premier coup. D'après Monsieur Schaerer, un problème de timing dans l'exécution des commandes du script en serait la cause, ce qui semble se confirmer, car une erreur différente apparaît à chaque lancement jusqu'à ce qu'il finisse par fonctionner. Le comportement de ce script semble donc non déterministe.

Lorsqu'on veut réexécuter le script alors qu'il a déjà été exécuté avec succès, il faut utiliser l'option *-r* en plus.

[92] La dernière version peut être trouvée ici : https://github.com/contiv/netplugin/releases

3.4.2 Exemple : Application avec deux containers

Maintenant que la base est installée, on va configurer notre cluster afin de pouvoir lancer 2 containers (Un serveur web et un serveur de base de données) sur un réseau commun avec les policies nécessaires pour permettre la communication à travers la fabric ACI.

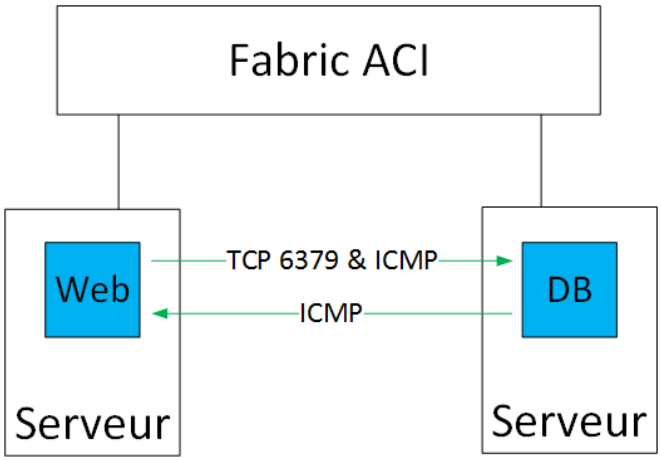


Figure 61 Schéma de l'exemple visé

3.4.2.1 Le fonctionnement de Contiv

Dans cet exemple, je n'utilise que le plugin réseau de contiv bien qu'il existe également un plugin pour les volumes ainsi que d'autres possibilités.

Comme nous l'avons vu dans la première partie de ce document, il existe des plugins permettant d'étendre les possibilités de Docker. En l'occurrence, le plugin réseau permet d'assigner un réseau de containers à des EPG et de définir des policies entre eux.

En pratique, dans cet exemple, un VLAN sera automatiquement attribué à chaque EPG afin que la fabric reconnaisse le trafic et le laisse transiter selon les policies configurées.
Pour déterminer dans quel EPG est un container, le plugin va simplement se baser sur le réseau spécifié au lancement du container comme nous le verrons plus loin.

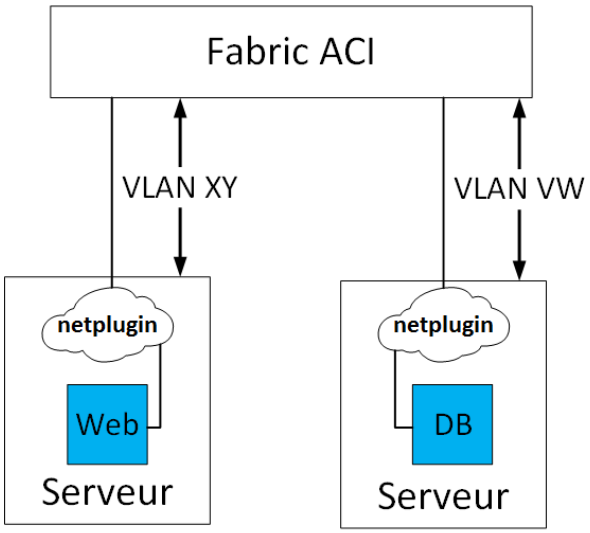


Figure 62 Les containers utilisent le plugin réseau et leur trafic est encapsulé dans un VLAN

Grâce à la commande netctl, nous pouvons contrôler le plugin réseau de contiv.
Tout d'abord, il faut indiquer au plugin la plage de VLAN à utiliser. En l'occurrence de 100 à 499 comme j'ai choisi arbitrairement dans la configuration de l'APIC tout à l'heure :

```
netctl global set --fabric-mode aci --vlan-range 100-499
```

Comme nous l'avons vu dans la deuxième partie de ce document, la notion de tenant est importante. Elle définit un client ou un département et les tenants sont isolés les uns des autres. Contiv reprend cette notion, il faut donc en déclarer un :

```
netctl tenant create mytenant
```

Au sein de ce tenant, il faut créer un réseau en spécifiant sa passerelle, le type d'encapsulation utilisé (ici des VLAN) et le nom souhaité :

```
netctl network create -t mytenant --s 10.25.0.0/24 --g 10.25.0.1 -e vlan
mytenant-contiv-net
```

Toutes les interactions étant basé sur des policies, il faut en définir au moins une au sein du tenant déclaré plus haut :

```
netctl policy create -t mytenant web2db
```

Une fois la policy créée, il faut lui attacher des end-point groups (EPG) :

```
netctl group create -t mytenant -p web2db mytenant-contiv-net db
```

1 Déclaration de l'EPG db

```
netctl group create -t mytenant -p web2db mytenant-contiv-net web
```

2 Déclaration de l'EPG db

Jusque-là, aucune règle n'a été ajoutée à la policy web2db. J'autorise le trafic entrant sur le port TCP 6379 en provenance de l'EPG web à l'aide de la commande suivante :

```
netctl policy rule-add -t mytenant -d in --protocol tcp --port 6379
--from-group web --action allow web2db 1
```

Il suffit, dès lors, de réunir tout ceci sous la forme d'un *application profile* pour que le plugin de Contiv répercute la configuration de l'APIC :

```
netctl app-profile create -t mytenant -g web,db mytenant-app-01
```

Le résultat est désormais visible dans l'APIC :

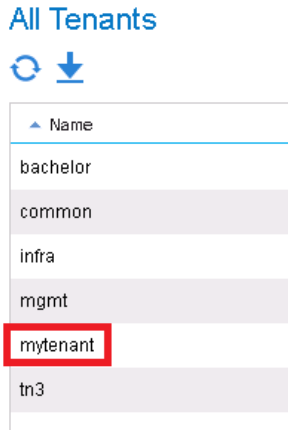


Figure 63 Le tenant est visible dans l'APIC

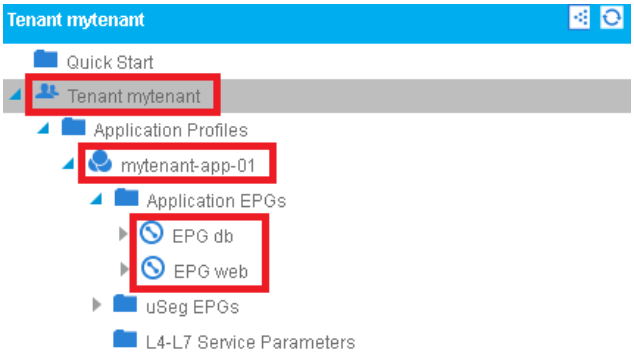
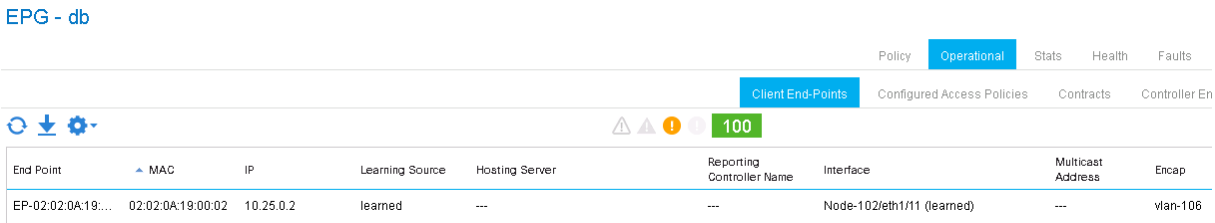


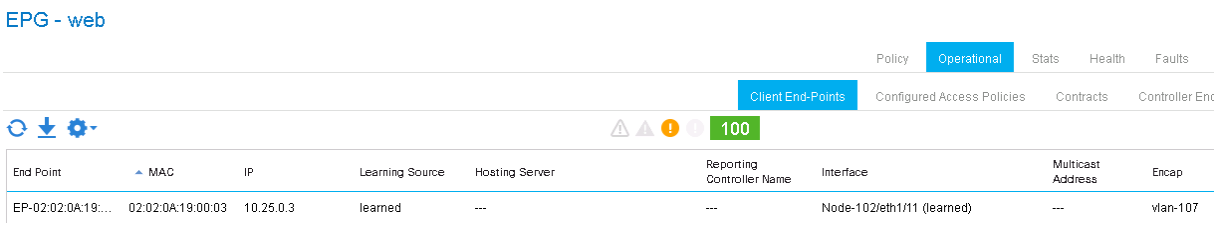
Figure 64 Le tenant contient l'application profile et les deux EPG

On peut ensuite lancer les containers en spécifiant le réseau dans lequel ils doivent se lancer ainsi que leur endpoint group :

```
docker run -itd --net web/mytenant --name mytenant-web01 jainvipin/web
docker run -itd --net db/mytenant --name mytenant-db01 jainvipin/redis
```

A ce stade, dans l'APIC, on voit que les containers sont bel et bien dans les EPG définis :





Pour le moment, le ping depuis le container web vers le container db ne devrait pas fonctionner puisqu'aucune policy allant dans ce sens n'a été configurée, essayons :

```
docker exec -it mytenant-web01 bash
```

```
root@aa9a2c43b897:/pycode# ping 10.25.0.3
PING 10.25.0.3 (10.25.0.3): 56 data bytes
^C--- 10.25.0.3 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
```

Figure 65 Le ping ne passe pas

On ajoute à ceci une deuxième règle pour autoriser le ping (ICMP) :

```
netctl policy rule-add -t mytenant -d in --protocol icmp --from-group
web --action allow web2db 2
```

En se rendant sur le GUI de l'APIC, il est possible de voir que la représentation graphique des policies :

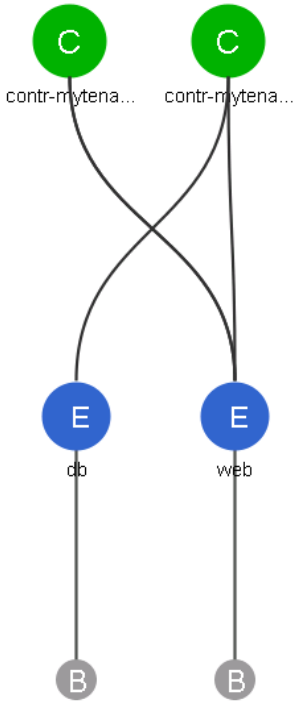


Figure 66 La vue des contrats entre les containers web et les containers db (icmp et port 6379)

On renouvelle le test :

```
docker exec -it mytenant-web01 bash
```

```
root@aa9a2c43b897:/pycode# ping 10.25.0.3
PING 10.25.0.3 (10.25.0.3): 56 data bytes
64 bytes from 10.25.0.3: icmp_seq=0 ttl=64 time=1.803 ms
64 bytes from 10.25.0.3: icmp_seq=1 ttl=64 time=0.206 ms
64 bytes from 10.25.0.3: icmp_seq=2 ttl=64 time=0.149 ms
^C--- 10.25.0.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
```

Figure 67 Le ping passe

De la même manière, l'application a désormais accès à la base de données. A ce stade, je ne peux malheureusement pas présenter de capture d'écran du résultat à cause d'un problème d'accès à l'infrastructure à distance lorsque je voulais documenter cette partie. Un problème d'alimentation électrique dans le labo de Cisco Zürich en est la cause.

3.4.3 Difficultés rencontrées

Cette troisième partie du travail n'a pas été sans difficulté. En effet, le projet Contiv est très mal documenté, le site officiel contient des exemples très/trop spécifiques qu'il est difficile de reproduire et la marche à suivre accompagnant ces exemples est toujours partielle.

Les informations sont, certes, présentes, mais réparties sur tout le site dans des chapitres qui ne sont que peu voire pas en lien avec le sujet. Lorsque j'ai écrit aux responsables du projet pour leur faire part de cette constatation, ils m'ont répondu qu'ils le savaient et qu'ils y travaillaient. Heureusement qu'ils sont assez réactifs pour répondre aux questions sur leur Slack[93].

Ce projet étant très jeune, il n'existe pas de documentation autre que la documentation officielle, il est donc difficile de trouver des informations ou des explications sur internet. C'est donc grâce à l'aide de Monsieur Rolf Schaerer que j'ai réussi à faire interagir Contiv avec l'APIC, mais même ensemble, il nous a fallu plusieurs jours pour en arriver à bout.

Le script d'installation de Contiv, par exemple, a demandé d'être relancé une dizaine de fois avant de fonctionner ce qui semble courant, car M. Schaerer a rencontré le même problème lorsqu'il a monté son propre labo.

[93] Slack est une plateforme de travail collaboratif : https://contiv.slack.com

4 Conclusion

A travers les différentes étapes de ce travail, il apparaît clairement que le monde du datacenter est en train de subir une évolution très importante.

Premièrement, du point de vue software, Docker a rendu les containers populaires et bien plus accessibles que ce qui existait auparavant. La baleine bleue apporte une approche radicalement différente de ce que nous connaissions jusque-là, la vague des microservices a très rapidement atteint les grands acteurs du monde de l'informatique et l'engouement qu'elle suscite ne passe pas inaperçu.
Au-delà de la flexibilité que Docker apporte en termes de *scalabilité*, il faut citer ses nombreux points forts tels qu'une sécurité accrue, un overhead quasiment nul (Moins de 1% selon Sebastien Pasche et Benoît Chalut de LeShop), la grande communauté qui gravite autour de ce projet open-source tant de grandes entreprises que des particuliers ou encore une documentation de qualité.
Il ne faut cependant pas s'aventurer n'importe comment dans les containers, il vaut mieux acquérir une solide expérience avec Docker avant d'essayer de l'implémenter en production sous peine de perdre la maîtrise de son infrastructure ou simplement de passer à côté des bénéfices qu'il peut apporter.

Deuxièmement, l'aspect réseau prend un tournant avec ACI. La technologie de Cisco répond totalement aux besoins apportés par la notion de cloud : flexibilité du réseau, sécurité renforcée par rapport au modèle traditionnel, facilité d'audit, intégration de services tiers agissant dans les couches L4 à L7 etc.
On assiste là à un changement de paradigme qui demandes aux équipes réseaux de se recentrer sur l'application plutôt que d'aborder en premier lieu les basses couches. Ce point de vue demande donc indubitablement de se remettre en question, car l'approche est ici également très différente de l'approche traditionnelle.
Lors de l'utilisation de systèmes ACI, il faut cependant accepter de ne pas avoir complètement la maîtrise de ce qui se passe dans la fabric, car ce modèle repose sur une sorte de confiance que doit avoir l'administrateur dans l'implémentation qui a été faite. Il contrôle ce qui se trouve dans sa périphérie, mais il doit pouvoir faire abstraction de ce qu'elle fait exactement, ce qui n'est pas chose aisée quand il s'agit de comprendre pourquoi telle ou telle configuration ne fonctionne pas.

A ce stade, la question qui se pose est : qu'en est-il de l'utilisation simultanée de Docker et de Cisco ACI ? Sur cet aspect, mes recherches me poussent à dire que c'est là que le bât blesse.
Dans un premier temps, lors de ce travail, ces technologies apparaissaient comme des solutions complémentaires, ACI allait probablement apporter une sécurité accrue à Docker ou tout autre avantage. En réalité, il est apparu qu'il fallait prendre le tout dans l'autre sens : grâce à Contiv, Docker peut fonctionner sur une infrastructure ACI sans qu'il y ait un autre intérêt pour Docker. Ce couplage a, au contraire, apporté une couche de difficulté supplémentaire.

En résumé, ACI est un produit très intéressant pour le datacenter en général, mais il n'apporte rien de particulier à Docker mis-à-part la compatibilité avec le fabric, ce qui est, bien entendu, nécessaire, ce serait une erreur de Cisco d'exclure un acteur aussi important que la baleine et ses containers de sa solution de réseau.
En revanche, Contiv, qui permet de faire le lien entre les 2 technologies, ne me paraît pas abouti. Les scripts qu'il fournit sont non-déterministe à tel point qu'il faut les lancer plusieurs fois avant qu'ils fonctionnent correctement et la documentation officielle est pour le moins désastreuse. Contiv semble donc être trop jeune pour être utilisé en production.

Finalement, il est important de ne pas oublier les bonnes pratiques du monde physique, ACI et Docker nous font, certes, changer de paradigme, mais il ne faut pas pour autant laisser de côté la notion de redondance.

4.1 Conclusion personnelle

Ce travail m'a permis de m'intéresser à des sujets qui m'étaient inconnus ou dont seul le nom m'était familier, ce qui était très stimulant. Ces sujets sont tellement vastes qu'ils pourraient être à l'origine de bon nombre de projets de semestre, de bachelor et de master, c'est d'ailleurs ce qui m'a empêché d'approfondir réellement les aspects techniques.
Il serait par ailleurs intéressant voire nécessaire, à l'avenir, de dédier, dans notre établissement, un cours à Docker et aux technologies qui sont employées par les containers tant ce sujet est incontournable autant pour les développeurs que les administrateurs réseau.

A titre personnel, j'ai trouvé à la fois intéressant mais également très challengeant de remettre en question la manière de concevoir les réseaux et les systèmes.
Au-delà de ça, les principales difficultés auxquels j'ai dû faire face concernaient le travail à distance que j'ai réalisé sur l'infrastructure mise à disposition par Cisco. En effet, il a fallu que je trouve un moyen d'installer mes machines à distance sans avoir un accès physique et en cas de problème, je ne pouvais pas faire de vérification ou de modification physique sur les machines sans avoir un intermédiaire, ce qui impliquait un délai d'attente. Je suis cependant reconnaissant envers Cisco pour ce qui m'a été mis à disposition et pour le temps qui m'a été consacré.

Pour finir, malgré ces différents challenges de qualité et ces sujets très intéressants, je dois admettre que j'ai particulièrement été frustré par la manière dont Contiv fonctionne, après avoir travaillé sur deux sujets très bien documentés, j'ai dû faire face à un projet peu utilisé avec une documentation insuffisante. Le seul moyen d'obtenir des réponses était de recevoir l'aide précieuse de Rolf Schaerer de Cisco ou de s'adresser directement aux développeurs sur le Slack[94] officiel. Cette partie du travail était donc difficile à aborder et le résultat n'est pas très impressionnant.

[94] Slack est un outil de travail participatif sous forme de chat avec différents canaux : https://contiv.slack.com

5 Annexes

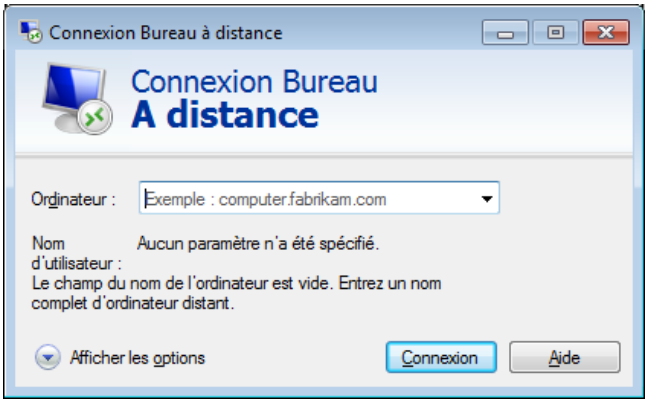
5.1 Utilisation d'une gateway RDP

Cette procédure a pour but de décrire les étapes à suivre pour se connecter en RDP à une machine en utilisant une gateway RDP comme j'ai dû le faire pour pouvoir accéder à distance à l'infrastructure mise à disposition par Cisco.

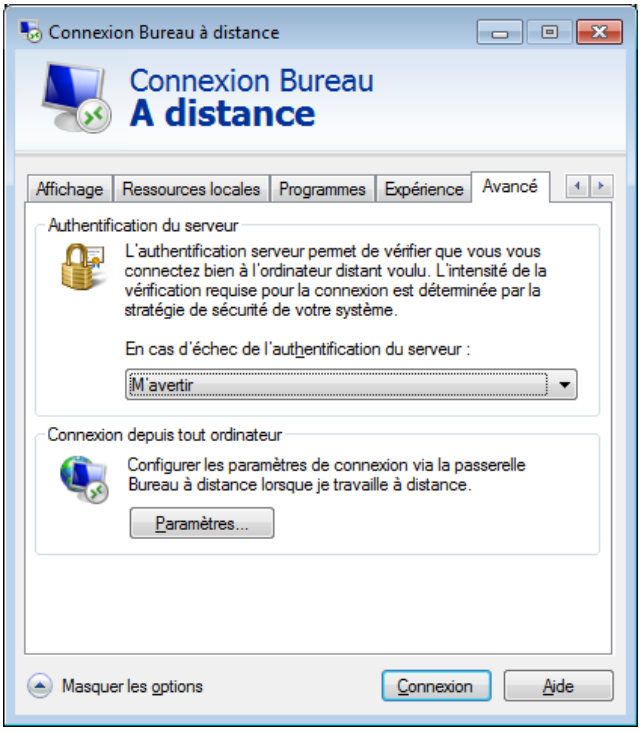
5.1.1 Windows

1) Sur Windows, utiliser le raccourci **Windows + r** pour ouvrir la boîte de dialogue « Exécuter »

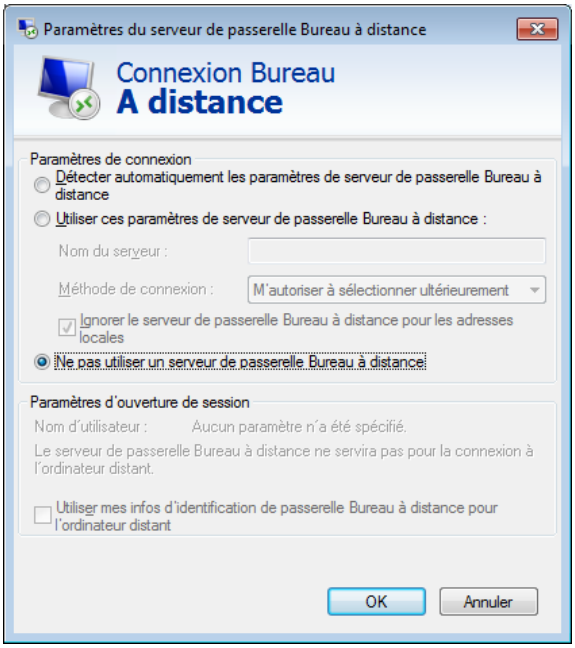
2) Saisir **mstsc** dans le champ et cliquez sur « Ok »



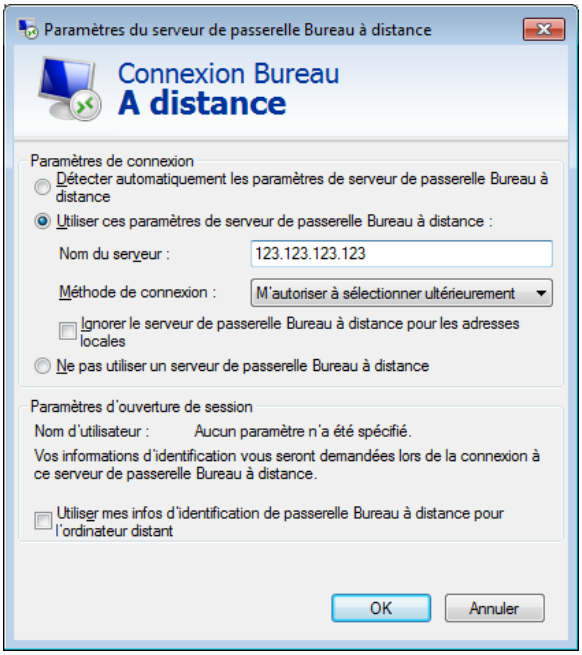
3) Cliquez sur « Afficher les options » et rendez-vous dans l'onglet « Avancé »



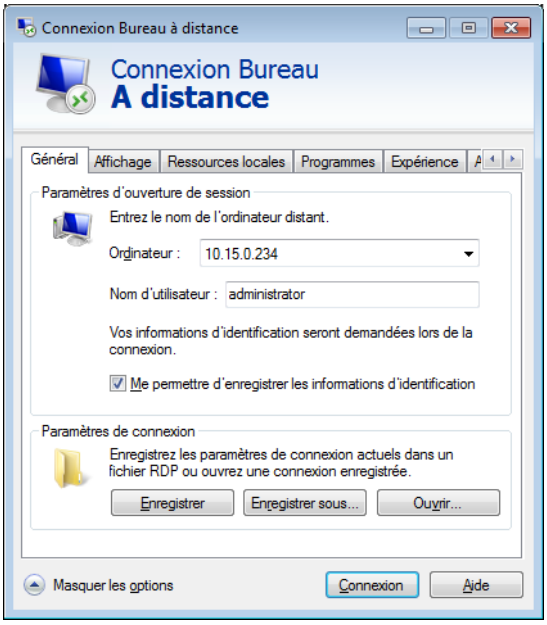
4) Cliquez sur le bouton « Paramètres »



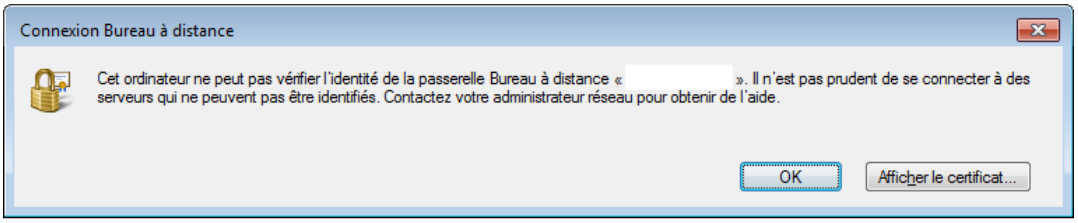
5) Cochez le deuxième radio bouton, renseignez l'adresse IP de la passerelle dans la case et décochez la case « Ignorer le serveur de passerelle […] pour les adresses locales »



6) Cliquez sur « Ok » et revenez à l'onglet général, entrez l'adresse du serveur RDP (généralement une adresse locale puisque le rôle de la passerelle est justement de séparer internet du réseau interne) et le nom d'utilisateur à utiliser.



7) Cliquez sur « Connexion », un message d'erreur concernant le certificat apparaîtra.

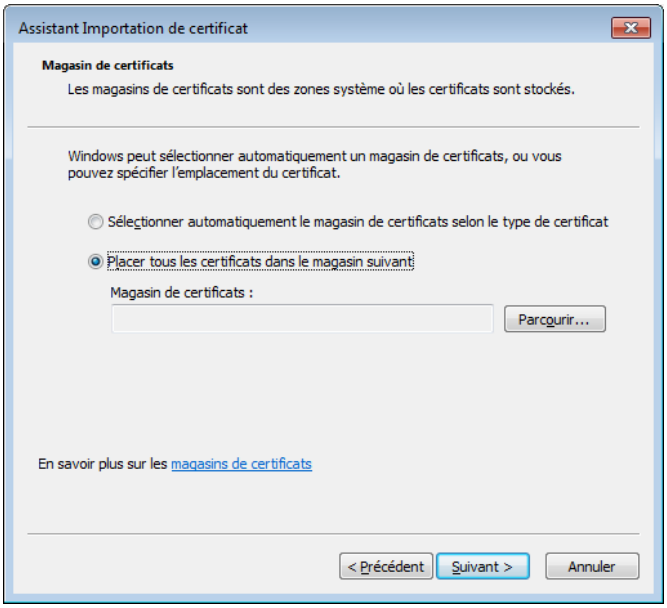


8) Cliquez sur « Affichez le certificat »

9) Dans l'onglet « Détails », cliquez sur « Copier dans un fichier », laissez les options par défaut et choisissez un emplacement pour le fichier.

10) Double-cliquez sur le fichier, cliquez sur « Installer le certificat »

11) Choisissez le magasin de certificats grâce au radio bouton



12) Cliquez sur « Parcourir » et choisissez « Autorités de certification racines de confiance »
Attention : Il est important de s'être assuré manuellement, si possible, que le certificat est bien celui de la passerelle, en particulier s'il est auto-signé. Installer un certificat comme autorité racine de confiance peut représenter un danger si la vérification n'est pas effectuée.

13) Validez et terminez l'opération. A la fin un avertissement apparaîtra, faites le contrôle nécessaire et cliquez sur « Oui »

14) Retourner sur le client RDP et réessayez de vous connecter. La première fois, les identifiants et mots de passe de la passerelle et du client sur le serveur vous seront demandés à tour de rôle.

5.1.2 Linux

Dans le monde Linux, les programmes clients RDP installés par défaut ne supportent généralement pas les passerelles RDP. Je conseille par conséquent l'utilisation de FreeRDP (http://www.freerdp.com).
La marche à suivre pour l'installation est disponible sur Github :
https://github.com/FreeRDP/FreeRDP/wiki/Compilation

Et voici la ligne de commande type pour se connecter en utilisant une passerelle RDP :

```
xfreerdp /v:AdresseDuServeur /u:UtilisateurServeur
/p:MotDePasseUtilisateur /g:AdresseDeGateway /gd:DomaineGateway
/gu:UtilisateurGateway /gp:MotDePasseGateway /size:90%
```

5.2 Installation à distance avec virtual KVM

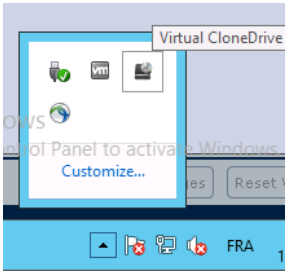
Cette procédure a pour objectif de décrire la marche à suivre permettant de monter un périphérique virtuel (en l'occurrence une image ISO) sur un serveur UCS à distance à l'aide du CIMC (Cisco Integrated Management Controller).

5.2.1 Montage d'une ISO sur une machine

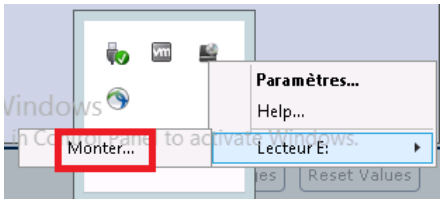
Dans un premier temps, il faut monter le fichier ISO du système d'exploitation à disposition sur une machine depuis laquelle on va faire l'installation. Dans le cas présent, la machine est un serveur Windows 2012 R2 auquel j'accède à distance via RDP.

Pour ce faire, il faut utiliser un programme permettant de créer un lecteur DVD virtuel sur lequel on va monter l'ISO. Je recommande le programme Virtual CloneDrive[95] developpé par la société Elaborate Bytes (une société suisse installée dans le canton de Zug) pour sa compatibilité avec toutes les versions récentes de Windows et sa simplicité d'utilisation.

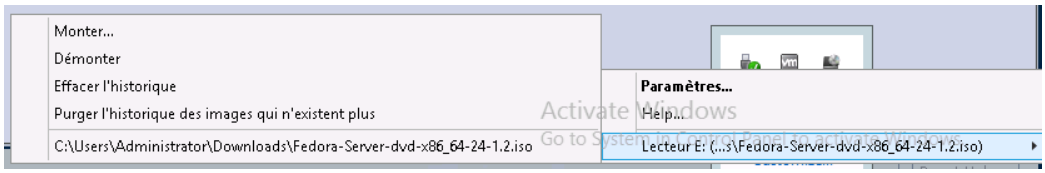
Une fois ce programme installé, il est visible dans la zone de notification de Windows :



Pour monter une image ISO, il suffit de faire un clic droit sur l'icône, descendre la souris sur le lecteur proposé (en l'occurrence E) et cliquer sur monter :



Une fois l'ISO sélectionnée, l'image est montée comme en témoigne la capture suivante :



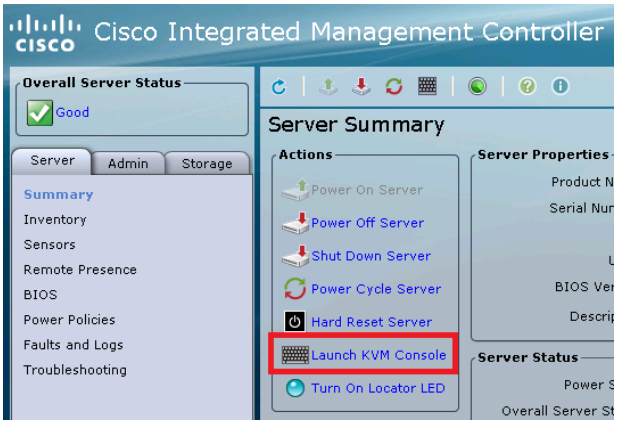
[95] Lien de téléchargement : https://www.elby.ch/products/vcd.html

5.2.2 Montage du périphérique virtuel sur le serveur

Afin de pouvoir installer Fedora sur les serveur UCS, il faut encore qu'ils aient accès au périphérique virtuel précédemment configuré.
Que le serveur soit une machine Cisco, HP ou autre, tous les constructeurs de serveurs professionnels sont équipés d'un système de management indépendant du système d'exploitation (si tant est qu'il y en ait un) permettant de configurer la machine à distance et d'y accéder. (virtual KVM[96], gestion des disques, températures, informations matérielles, gestion du BIOS etc.).
En l'occurrence, la solution de Cisco se nomme CIMC (Cisco Integrated Management Controller) dont le KVM virtuel permet d'ajouter des « Medias virtuels ».

1) Accédez au GUI du CIMC en saisissant https://addresse-de-management-du-serveur dans votre navigateur.

2) Cliquez sur le lien pour ouvrir la console KVM

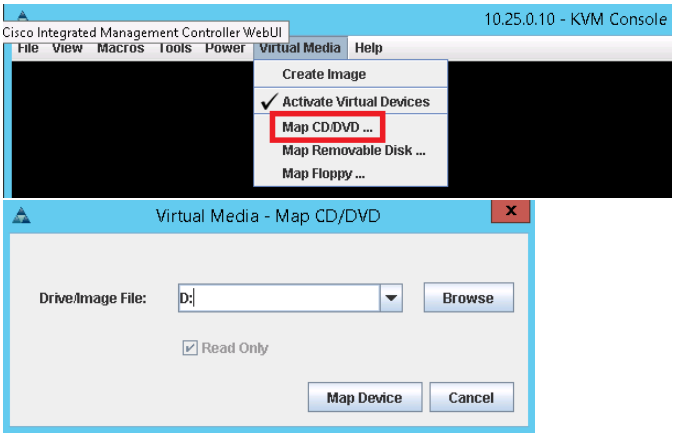


3) Après avoir lancé l'exécutable Java, activez les périphériques virtuels

4) Accepter l'avertissement (La session ne sera pas chiffrée, si vous n'êtes pas dans un milieu sécurisé, cela peut représenter un danger.)

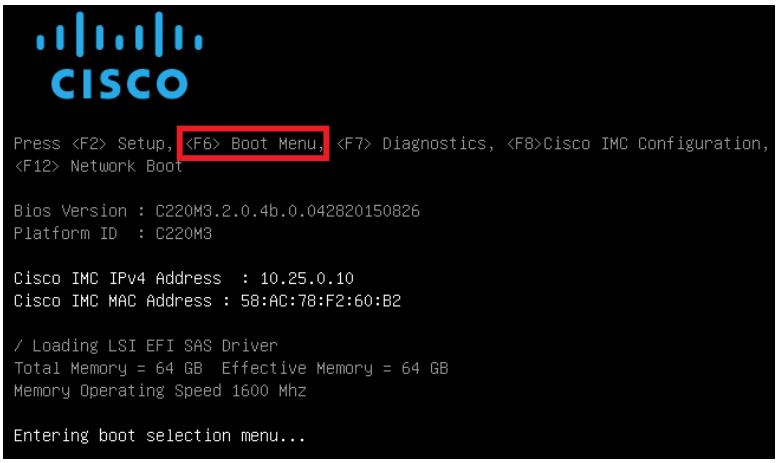
[96] KVM = Keyboard Video Mouse

5) Mappez ensuite le lecteur



6) Redémarrez la machine (Menu « Power » du KVM)

7) Rentrez dans le menu de démarrage (F6 dans ce cas)



8) Il ne reste plus qu'à choisir le disque mappé précédemment pour démarrer dessus