DÉPLOYER ET GÉRER VOS HÔTES DOCKER AVEC DOCKER MACHINE

Introduction

Docker Machine est un **outil de provisioning et de gestion des hôtes Docker** (hôtes virtuels exécutant le moteur Docker). Vous pouvez utiliser Docker Machine pour créer des hôtes Docker sur votre ordinateur personnel ou sur le datacenter de votre entreprise à l'aide d'un logiciel de virtualisation tel que VirtualBox ou VMWare, vous pouvez aussi déployer vos machines virtuelles chez des fournisseurs de cloud, tels que Azure, AWS, Google Compute Engine, etc ..

À l'aide de la commande `docker-machine`, vous pouvez démarrer, inspecter, arrêter et redémarrer un hôte géré ou mettre à niveau le client et le moteur Docker et configurer un client Docker pour qu'il puisse communiquer avec votre hôte. En bref il **crée automatiquement des hôtes Docker**, y **installe le moteur Docker**, puis **configure les clients docker**.

Installation de Docker Machine

Voici la **commande qui permet d'installer Docker Machine sous Linux**.

```
base=https://github.com/docker/machine/releases/download/v0.16.0 &&
curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Pour pouvoir **activer l'auto-completion des commandes Docker Machine**, il suffit de créer un script qu'on va nommer **`docker-machine-prompt.bash`** dans le dossier **`/etc/bash_completion.d`** ou dans le dossier **`/usr/local/etc/bash_completion.d`** et de

coller dedans le contenu ci-dessous :

```
sudo nano /etc/bash_completion.d/docker-machine-prompt.bash
```

```
base=https://raw.githubusercontent.com/docker/machine/v0.16.0
for i in docker-machine-prompt.bash docker-machine-wrapper.bash docker-machine.bash
do
    sudo wget "$base/contrib/completion/bash/${i}" -P /etc/bash_completion.d
done
```

Enfin, il faut lancer la commande `source` pour charger votre script d'auto-completion :

```
source /etc/bash_completion.d/docker-machine-prompt.bash
```

Pour ceux qui ont installé Docker sur une **machine Windows** pro avec HyperV d'activé, il n y'a pas besoin d'installation car Docker machine est installé par défaut.

Découverte des drivers et des commandes Docker Machine

Docker machine utilise le concept des **drivers** (en fr : pilotes). Les drivers vous permettent depuis votre Docker machine. de créer un ensemble complet de ressources sur vos machines virtuelles sur des services tiers tels qu'Azure, Amazon, VirtualBox, etc. Vous retrouverez la liste des différents drivers ici.

Avant de vous décrire l'utilisation de certains drivers. Voici d'abord **la commande qui permet de créer une machine virtuelle depuis votre Docker Machine** :

```
docker-machine create --drive <DRIVER NAME> <MACHINE NAME>
```

La commande `docker-machine create` **télécharge une distribution Linux légère nommée boot2docker** venant avec le moteur Docker installé et crée et démarre la

machine virtuelle. Les options de cette commande peuvent différer selon le type de driver que vous utilisez.

Nous allons voir ci-dessous comment créer des hôtes Docker onpremise avec le driver virtualBox et hyperv mais aussi dans le Cloud avec le driver d'AWS (Amazon Web Service) nommé "amazonec2".

Découverte du pilote VirtualBox et utilisation des commandes Docker Machine

Je suis actuellement sous Linux avec la distribution **Fedora 30**. Si vous êtes sous une autre distribution alors la configuration risque d'être un peu différente. Dans tous les cas, voici la **configuration requise pour le driver VirtualBox** :

- Virtualbox à partir de la version 5
- Le module de noyau vboxdrv

Je vous fais confiance pour l'installation de VirtualBox, mais si jamais vous rencontrer des problèmes, alors n'hésitez pas à m'en faire part dans l'espace commentaire, il est prévu pour ça . En ce qui me concerne j'ai téléchargé VirtualBox en version 6.0.10.

Pour **installer le module de noyau vboxdrv**, il faut au préalable **installer le package kernel-devel**. Pour information le package kernel-devel télécharge les fichiers d'en-têtes du noyau Linux qui vont permettre aux développeurs d'accéder aux différentes fonctionnalités du noyau. De façon plus simple, il est nécessaire au développement et à la compilation de pilotes. Et c'est exactement ce qu'il nous faut !

```
sudo dnf -y install kernel-devel
```

> **Attention**
>
> Pour ceux qui sont sous une autre distribution, veuillez chercher l'équivalent de ce package sur votre distribution.

Lancez ensuite la commande suivante pour **installer le module vboxdrv** :

```
sudo /sbin/vboxconfig
```

Résultat :

```
vboxdrv.sh: Stopping VirtualBox services.
vboxdrv.sh: Starting VirtualBox services.
vboxdrv.sh: Building VirtualBox kernel modules.g
```

Une fois les deux prérequis de configurations satisfaites, vous pouvez dès lors créer votre hôte Docker en lançant la commande `create` en utilisant le driver `virtualbox` avec les options par défaut :

```
docker-machine create --driver virtualbox vbox-test
```

Résultat :

```
Running pre-create checks...
(vbox-test) Image cache directory does not exist, creating it at /home/hatim/.docker/
(vbox-test) No default Boot2Docker ISO found locally, downloading the latest release
(vbox-test) Latest release for github.com/boot2docker/boot2docker is v18.09.7
(vbox-test) Downloading /home/hatim/.docker/machine/cache/boot2docker.iso from https
(vbox-test) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtua
```

Ensuite, **vérifiez la liste des machines Docker disponible** en exécutant la commande suivante :

```
docker-machine ls
```

Résultat :

```
NAME        ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKE
vbox-test   -        virtualbox   Running   tcp://192.168.99.100:2376           v18.0
```

D'après le résultat, notre hôte `vbox-test` est bien présent avec l'état `Running` et possède le moteur docker en version v18.09.7.

Si vous retournez à la fin du résultat de la commande `docker-machine create`, vous remarquerez le message suivant (traduit en français) : *"Pour voir comment connecter Docker à cette machine, exécutez: docker-machine env vbox-test"*. Cette manipulation, va nous permettre de **récupérer les variables d'environnements de la nouvelle VM à exporter**. Parfait, utilisons la alors :

```
docker-machine env vbox-test
```

Résultat :

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/home/hatim/.docker/machine/machines/vbox-test"
export DOCKER_MACHINE_NAME="vbox-test"
# Run this command to configure your shell:
# eval $(docker-machine env vbox-test)
```

Le résultat nous indique clairement que si on souhaite **utiliser le moteur Docker de la machine virtuelle sur notre shell courant** il faut alors utiliser la commande suivante :

```
eval $(docker-machine env vbox-test)
```

En exécutant cette commande sur votre shell courant, alors n'importe quelle commande Docker que vous exécuterez, sera dorénavant directement prise en compte par votre hôte Docker `vboxt-test` et non plus par votre hôte maître.

Par ailleurs si vous souhaitez **vérifier sur quelle hôte Docker se lanceront vos prochaines commandes docker** alors soit vous vérifiez si une étoile existe dans la colonne `ACTIVE` de la commande `docker-machine ls`. Soit plus simple encore, vous lancez la commande suivante :

```
docker-machine active
```

Résultat :

```
vbox-test
```

Le résultat nous indique distinctement, que nos futurs commandes docker sur le shell courant s'exécuteront directement sur la machine Docker `vbox-test`.

Afin de vous prouver que c'est effectivement le cas, je vais télécharger et exécuter l'image httpd sur le shell courant :

```
docker run -d -p 8000:80 --name vbox-test-httpd htppd
```

À présent, ouvrez un nouveau terminal et vérifiez les conteneurs disponibles, vous verrez ainsi que vous ne retrouverez pas le conteneur `vbox-test-httpd` créé précédemment :

```
docker ps
```

Résultat :

```
CONTAINER ID    IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
```

Cependant si je retourne sur mon ancien shell avec la machine docker `vbox-test` activée, dans ce cas j'obtiendrai bien un résultat avec le conteneur `vbox-test-httpd` :

```
docker ps
```

Résultat :

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STA
347723c8291f        httpd               "httpd-foreground"  3 minutes ago       Up
```

Vous n'êtes pas encore convaincu ? Alors **connectez vous carrément à la machine Docker** à l'aide de la commande suivante :

```
docker-machine ssh vbox-test
```

Résultat :

```
   ( '>')
  /) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
 (/-_--_-\)           www.tinycorelinux.net

docker@vbox-test:~$
```

Revérifiez une nouvelle fois la liste de vos conteneurs et vous verrez que le conteneur `vbox-test-httpd` est bien dedans :

```
docker ps
```

Résultat :

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STA
347723c8291f        httpd               "httpd-foreground"  3 minutes ago       Up
```

Pour vous **assurer que le client Docker est automatiquement configuré au début de chaque session de shell**, vous pouvez alors intégrer la commande `eval $(docker-machine env vbox-test)` votre fichier `~/.bash_profile`.

Si vous pensez avoir fini d'utiliser une machine Docker, vous pouvez l'**arrêter** avec la commande `docker-machine stop` et la **redémarrer** plus tard avec la commande `docker-machine start`, example :

```
docker stop vbox-test
```

```
docker start vbox-test
```

Enfin, vous pouvez surcharger les ressources allouées automatiquement par défaut à hôte Docker en utilisant les options venant avec le driver virtualbox. Dans cet exemple je vais créer une machine Docker avec 30 Go d'espace disque (20 Go par défaut) et avec 2 Go de ram (1Go par défaut) et

```
docker-machine create -d virtualbox \
--virtualbox-disk-size "30000" \
--virtualbox-memory "4000" \
vbox-test-bigger
```

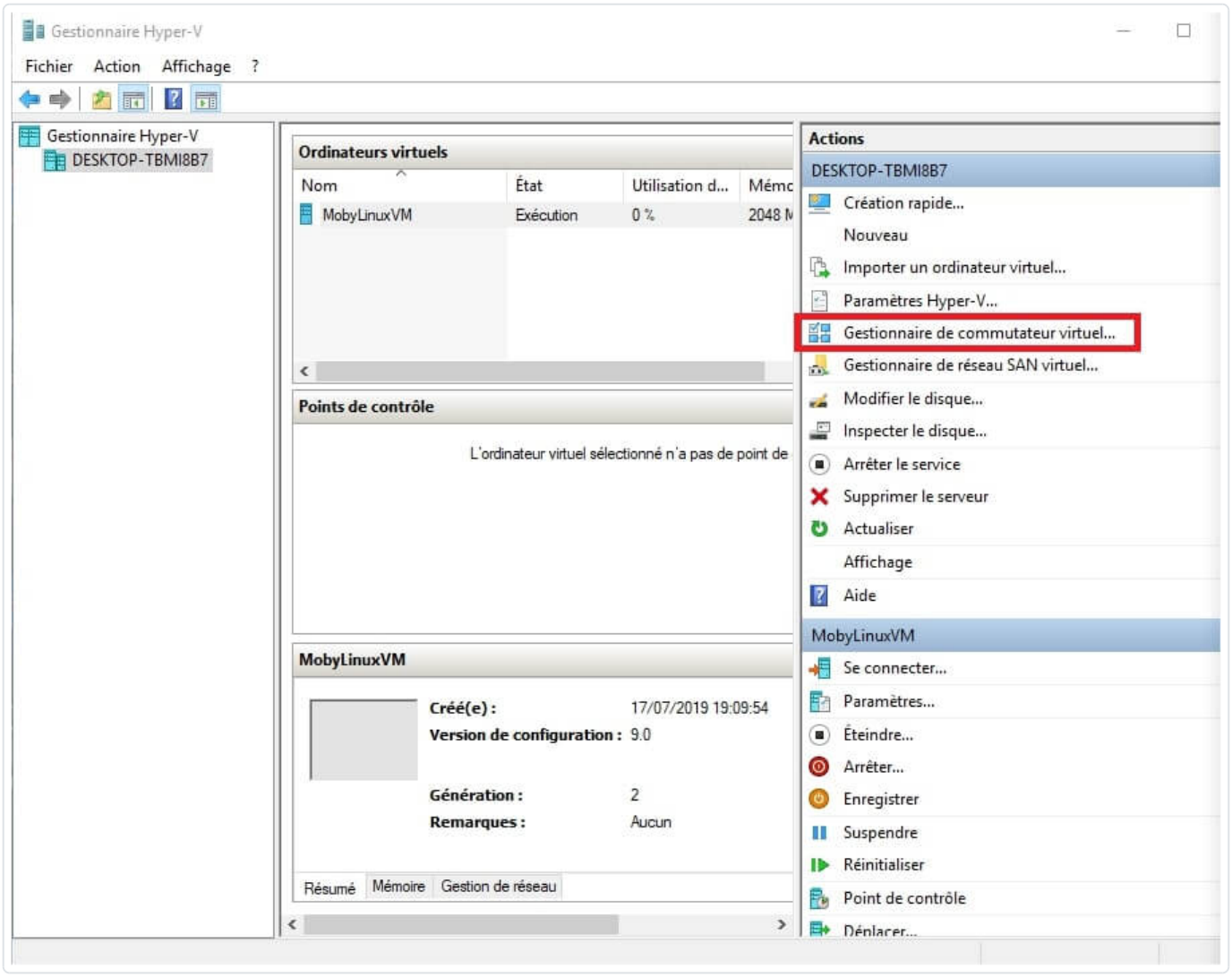
Vous retrouverez plus d'informations sur les options liées au driver virtualbox ici.

Le pilote HyperV

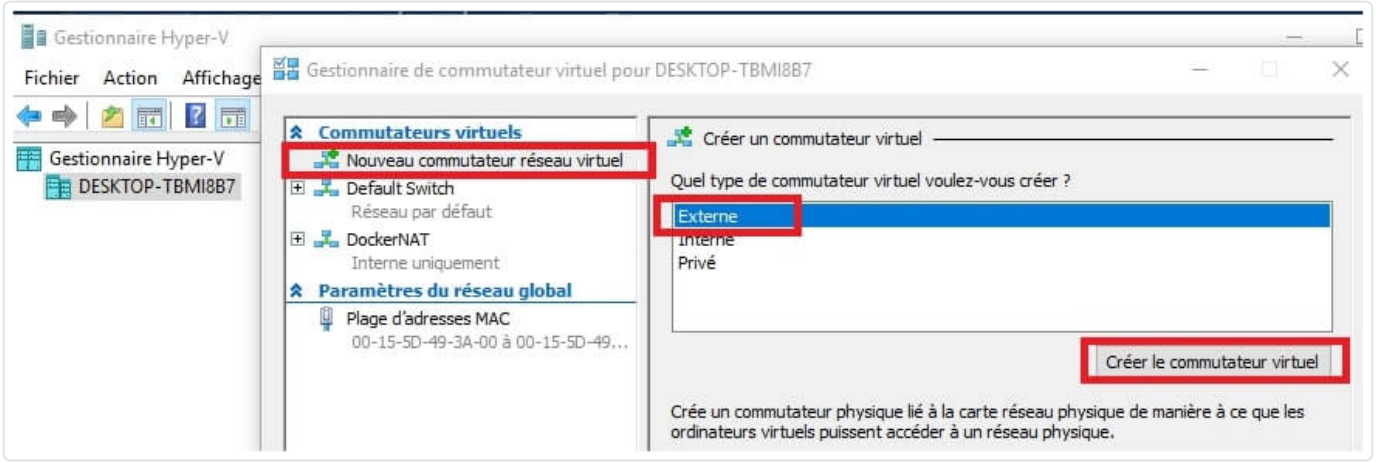
À cet instant je suis passé sur ma machine Windows afin d'utiliser le pilote hyperv et par la suite créer une machine docker basée sur ce driver.

Si vous avez déjà un switch réseau externe, ignorez cette configuration et allez voir directement la commande utilisant le driver hyperv. Mais si ce n'est pas le cas alors il faut en créer un en suivant les instructions suivantes :

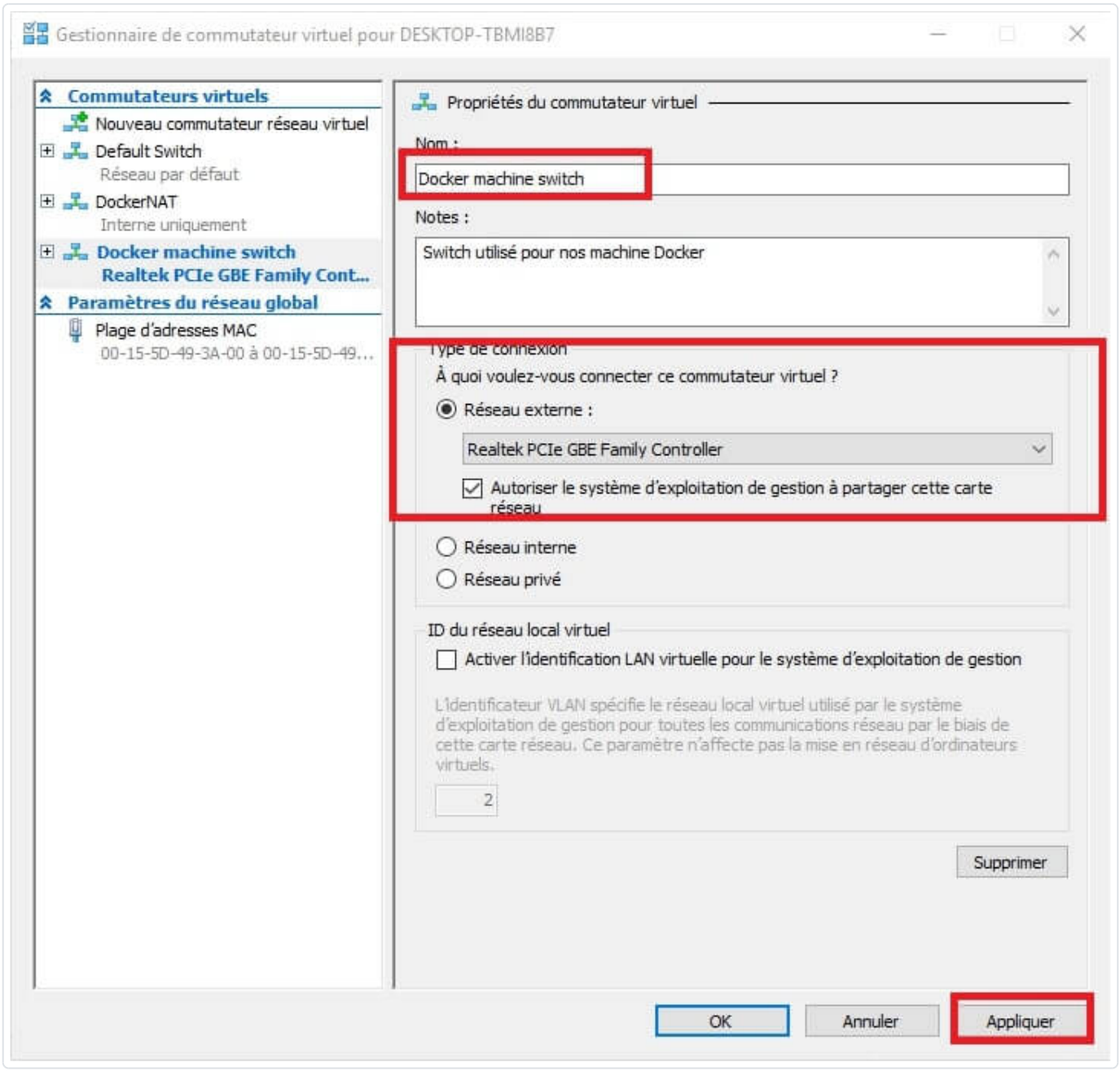
Ouvrez le gestionnaire Hyper-V et sélectionnez le "Gestionnaire de commutateur virtuel" dans le panneau d'actions de droite :



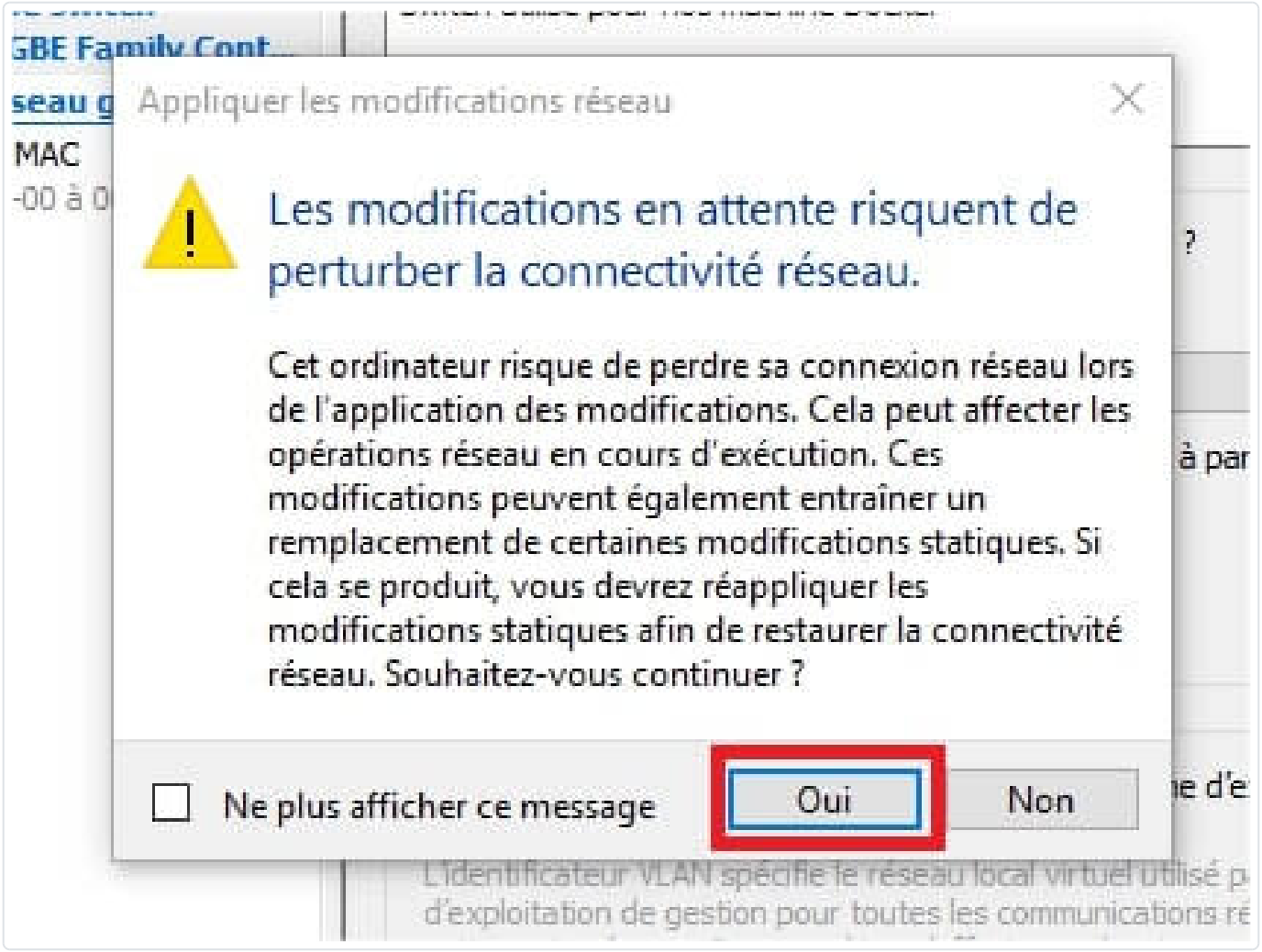
Ensuite, **configurez un nouveau switch réseau externe** à utiliser à la place du switch réseau DockerNAT :



Pour cet exemple, nous allons créer un switch virtuel appelé "Docker machine switch".



Ignorer l'avertissement en appuyant sur le bouton "oui".



Une fois le switch externe créé, on peut enfin l'utiliser pour déployer notre machine Docker depuis le driver hyperv. Lancez un powershell en mode administrateur et exécuter la commande suivante :

```
docker-machine create --driver=hyperv --hyperv-virtual-switch "Docker machine switch"
```

Résultat :

```
Running pre-create checks...
Creating machine...
(hyperv-test) Copying C:\Users\hatim\.docker\machine\cache\boot2docker.iso to C:\Use
(hyperv-test) Creating SSH key...
(hyperv-test) Creating VM...
(hyperv-test) Using switch "Docker machine switch"
...
Checking connection to Docker...
Docker is up and running!
```

```
To see how to connect your Docker Client to the Docker Engine running on this virtua
```

Vérifions ensuite la liste des hôtes Docker :

```
docker-machine ls
```

Résultat :

```
NAME          ACTIVE   DRIVER   STATE     URL                         SWARM   DOCKER
hyperv-test   -        hyperv   Running   tcp://192.168.0.19:2376             v18.09.7
```

Nous voyons bien notre nouvelle machine Docker `hyperv-test` avec le moteur Docker en version v18.09.7.

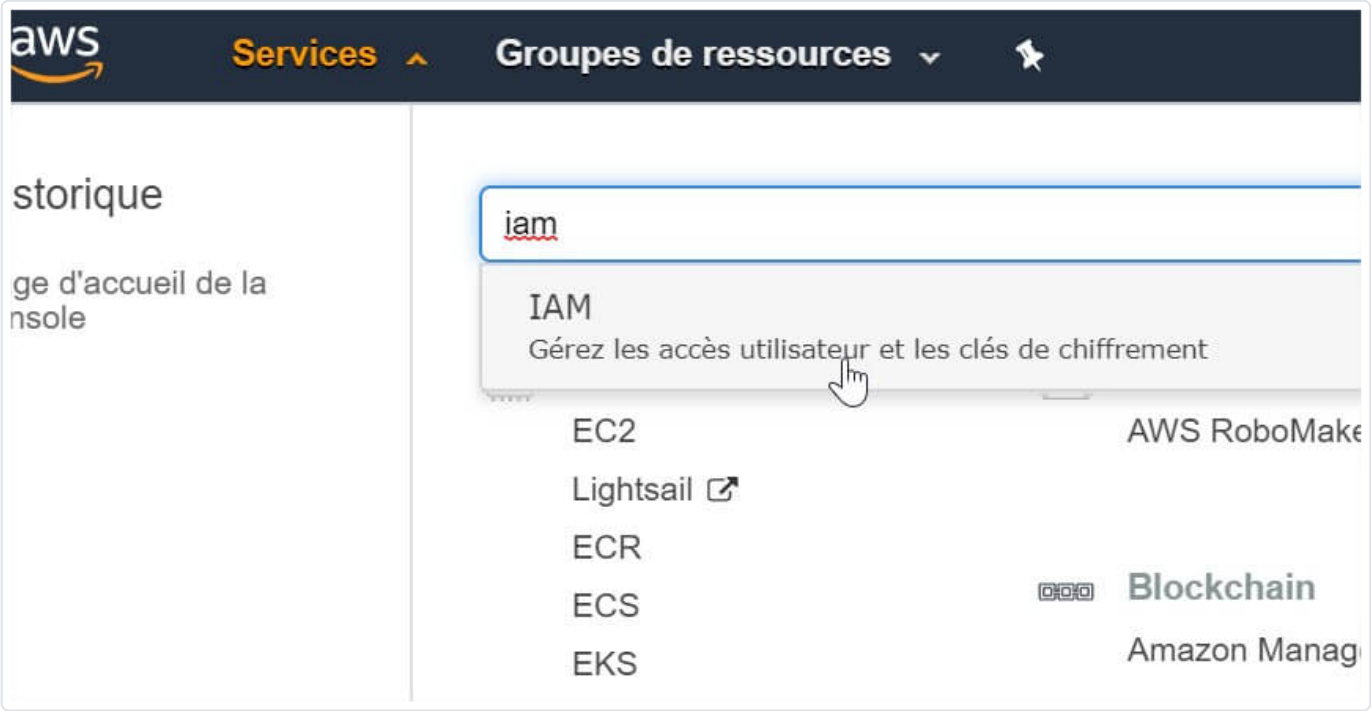
Le pilote amazonec2 (cloud)

Pour changer un peu du déploiement local, nous utiliserons cette fois-ci un service cloud, plus précisément nous utiliserons le driver amazonec2 qui va nous permettre de créer des machines sur le service cloud AWS (Amazon Web Services).

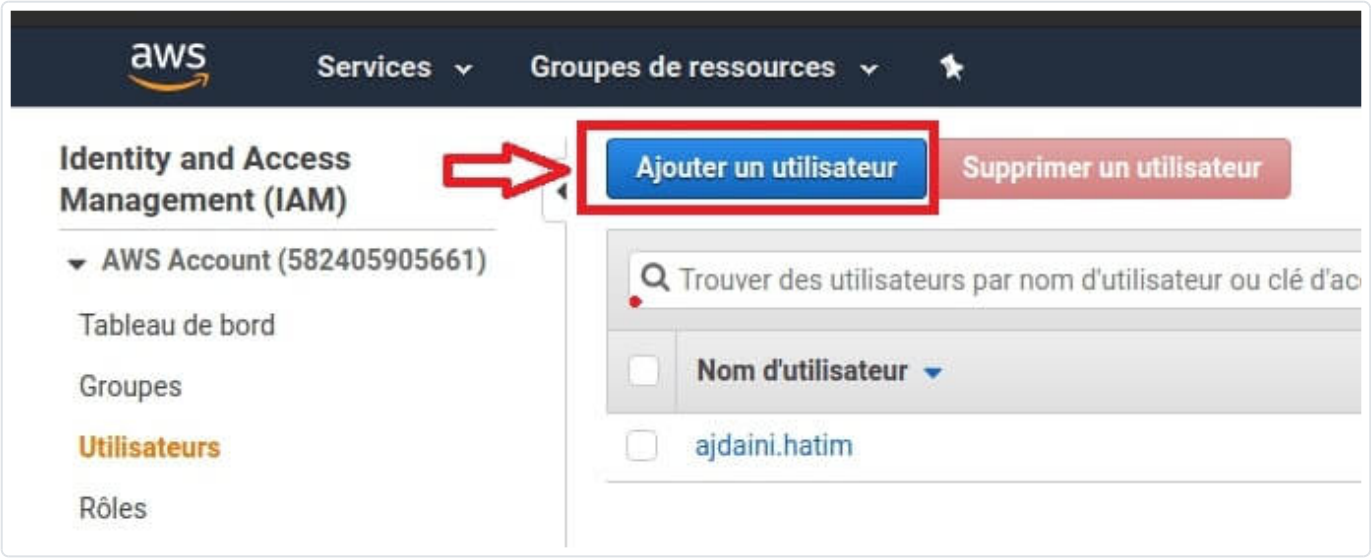
Pour créer des machines sur AWS , vous devez fournir deux paramètres :

- un ID de clé d'accès AWS
- une clé d'accès secrète AWS

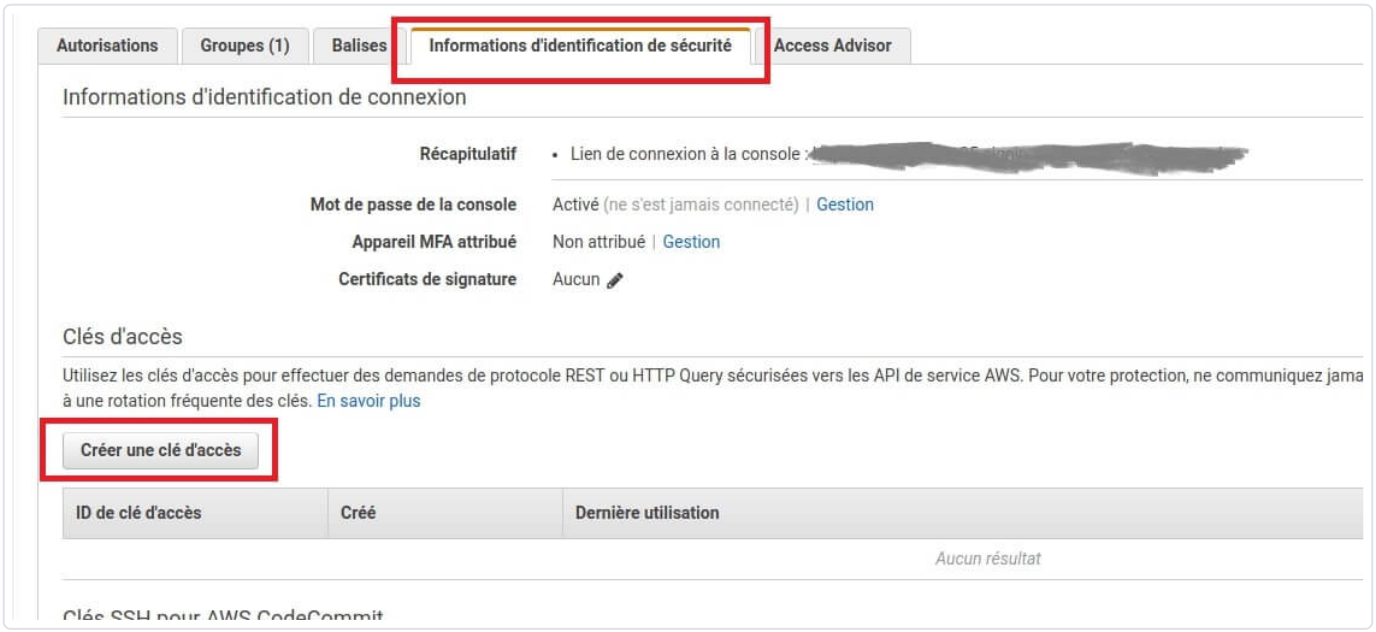
Rendez-vous dans le service IAM depuis votre console AWS :



Une fois rendu dans le service IAM, si vous n'avez pas encore d'utilisateur, créez-en un en appuyant sur le bouton "Ajouter un utilisateur" :



Après cela, cliquez sur le bouton "Créer une clé d'accès" :



Récupérer maintenant votre ID et clé d'accès secrète de votre compte AWS :

Configurer les informations d'identification en utilisant le fichier d'informations d'identification standard du fichier Amazon AWS **~/.aws/credentials**, de sorte que vous n'ayez plus besoin de les saisir à chaque fois que vous exécutez la commande `create`. Voici un exemple du fichier d'identification :

```
[default]
aws_access_key_id = AKIAYPGQYET63DHEKXKA
aws_secret_access_key = VOTRE-CLE-SECRETE
```

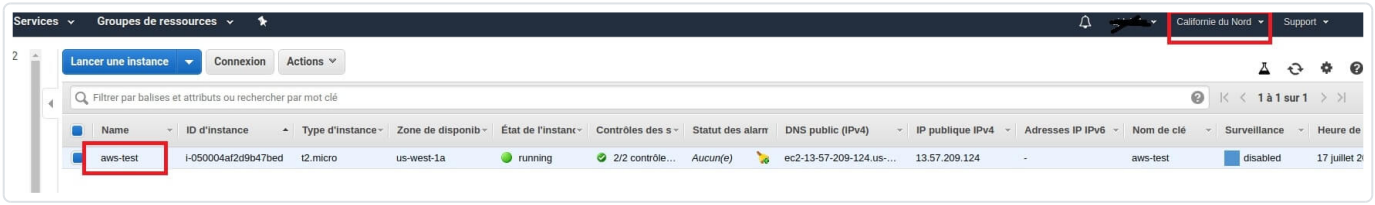
On peut dès à présent créer notre instance EC2 (VM aws) depuis notre machine maître Docker. Dans cet exemple nous allons utiliser la région us-west-1 et autoriser le port 8000 dans le security groupe (firewall AWS) lié à l'instance EC2 :

```
docker-machine create --driver amazonec2 --amazonec2-open-port 8000 --amazonec2-regi
```

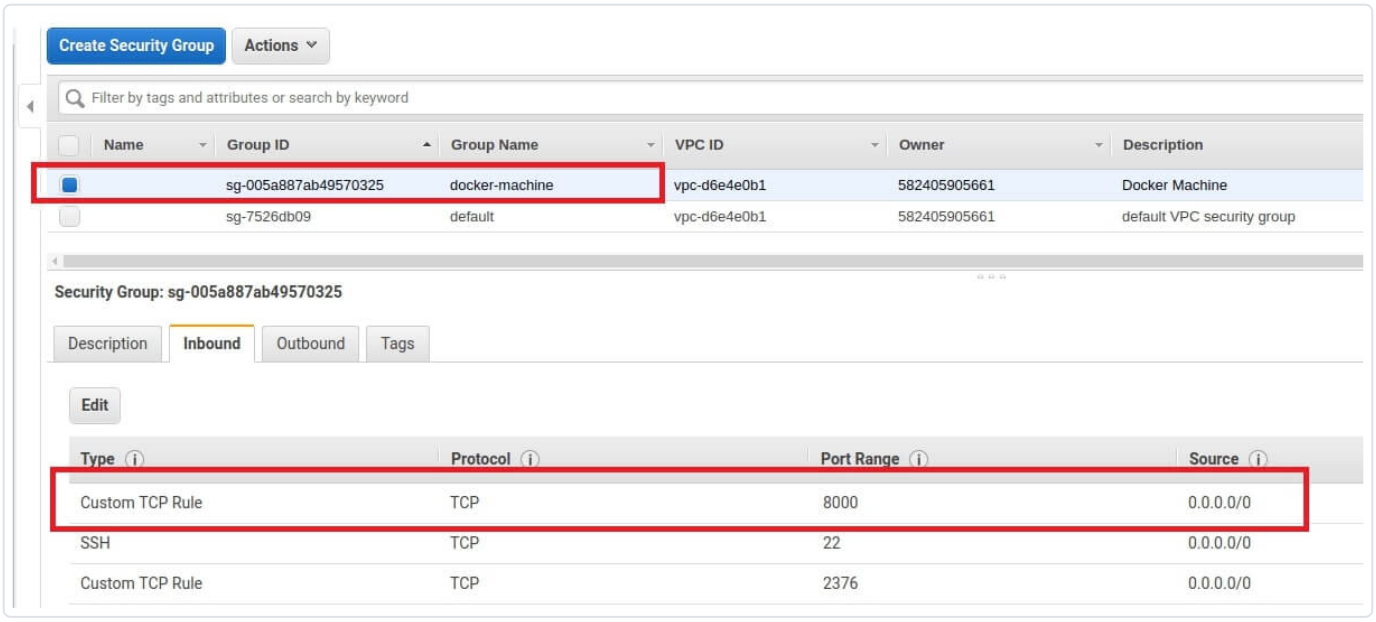
Résultat :

```
Running pre-create checks...
Creating machine...
(aws-test) Launching instance...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtua
```

Nous allons vérifier subséquemment sur notre console si notre instance EC2 est bien présente dessus. Pour cela, rendez-vous dans le service EC2 et assurez-vous d'être bien dans la même région que celle lancée dans la commande `create` :



Cool, notre instance EC2 `aws-test` est bien présente. Maintenant vérifions si le port 8000 est bien autorisé dans le security group lié à cette instance :



Le security group autorise bel et bien le port 8000, vous pouvez dès à présent lancer vos conteneurs directement sur votre instance EC2 depuis votre machine maître. Dans cet exemple nous allons instancier l'image httpd dans notre nouvelle machine Docker `aws-test` .

Premièrement, nous allons rendre notre hôte Docker `aws-test` active :

```
eval $(docker-machine env aws-test)
```

Deuxièmement, nous allons télécharger et exécuter notre image httpd

```
docker run -d -p 8000:80 --name httpdc httpd
```

Si vous visitez la page http://VOTRE_IP:8000, vous observerez alors le message "It works!".

Supprimer vos machines Docker

Comme je n'ai plus besoin de mes machines, je peux alors les supprimer. Pour ce faire, je vais utiliser la commande `docker-machine rm <MACHINE_NAME>`. Cette commande aura pour effet de **supprimer définitivement la machine Docker** de votre

plateforme de gestion de virtualisation locale mais aussi de la supprimer de votre fournisseur de cloud, si jamais vous en utilisez un.

```
docker-machine rm -f aws-test
```

```
docker-machine rm -f vbox-test
```

Conclusion

Nous avons utilisé Docker Machine pour créer des hôtes Docker localement mais aussi dans le Cloud, cela nous montre à quel point il est facile de déployer et des machines Docker n'importe où et de centraliser la gestion de ces VMs depuis une seule machine maître. Comme à mon habitude, je partage avec vous un **aide-mémoire résumant les différentes commandes de Docker Machine**.

```
## Créer une machine Docker
docker-machine create -d <DRIVER NAME> <MACHINE NAME>
    -d ou --driver : choisir un driver

## Rendre une machine Docker active
eval $(docker-machine env <MACHINE NAME>)

# Lister les machines Docker
docker-machine ls

# Vérifier quelle est la machine Docker active dans le shell courant
docker-machine active

## Supprimer un ou plusieurs machine(s) Docker
docker-machine rm <MACHINE NAME>
    -f ou --force : forcer la suppression

## Se connecter en ssh sur une machine Docker
docker-machine ssh <MACHINE NAME>

## Stopper une machine Docker
docker-machine stop <MACHINE NAME>

## Démarrer une machine Docker
docker-machine start <MACHINE NAME>
```

```
## Redémarrer une machine Docker
docker-machine restart <MACHINE NAME>

## Récolter des informations sur une machine Docker
docker-machine inspect <MACHINE NAME>

## Récupérer les variables d'environnements d'une machine Docker
docker-machine env <MACHINE NAME>

## Mettre à niveau une machine Docker vers la dernière version de Docker
docker-machine upgrade <MACHINE NAME>
```