CoreOS: Container Linux overview

Table of Contents

Habituellement, pour fabriquer une VM, on procède ainsi :

1. on installe l'OS dans une VM,
2. on fait une image de cette dernière,
3. on clône cette image.

Le choix d’un OS pour une VM se limite souvent à ceux que l'on utilise pour les serveurs physiques : Ubuntu, RHEL, CentOS.

Mais les VM dans le cloud ne sont pas comme des serveurs « normaux » : il peut y en avoir beaucoup, elles sont très volatile. On voit donc que les vieux outils ne sont plus très adaptés à ces OS tournant dans des datacenters, surtout lorsqu'on veut une architecture rapidement scalable.

Et c'est ici qu'arrive CoreOS.

Basé sur Chrome OS la première image a été diffusée le 25/07/2013. Il a pour but d'être un OS taillé pour les VM, et spécialement conçu pour Docker (qui est inclus d'office dans la distribution)

En quelques mots, CoreOS est :

- **Minimaliste**: ne sont installés que les composants nécessaires pour faire tourner les containers pour les applications.
- **Fiable**: Les mises à jour se font automatiquement.

CoreOS peut tourner sur n'importe quelle plateforme de virtualisation et fournisseur cloud. Tout se fait par le réseau.

Installation et configuration

Initialisation via PXE

Ces instructions décrivent l'amorçage de `Container Linux` via PXE sur du matériel réel ou virtuel. Par défaut, `Container Linux` fonctionnera complètement en RAM.

Un minimum de 2 Go de RAM est requis pour démarrer `Container Linux` via PXE.

Configurer pxelinux

On suppose qu'un serveur PXE en état de marche utilisant pxelinux est installé et configuré.

Lors de la configuration du `Container Linux` , certaines options du noyau dans `pxelinux.cfg` peuvent être utiles, mais elles sont toutes facultatives.

- `rootfstype=tmpfs`: Utilise `tmpfs` pour le système de fichiers racine accessible en écriture. Ceci est le comportement par défaut.
- `rootfstype=btrfs`: utilise `btrfs` dans la RAM pour le système de fichiers racine inscriptible. Le système de fichiers consomme plus de RAM au fur et à mesure de sa croissance, jusqu'à un maximum de 50%. La limite n'est pas configurable pour le moment.
- `root`: utilise un système de fichiers local pour root au lieu de l'une des deux options in-ram ci-dessus. Le système de fichiers doit être formaté (peut-être en utilisant Ignition) mais peut être complètement vide; il sera initialisé au démarrage. Le système de fichiers peut être spécifié par l'un quelconque des moyens usuels, y compris périphérique, étiquette ou UUID; Par exemple, `root=/dev/sda1`, `root=LABEL=ROOT` ou `root=UUID=2c618316-d17a-4688-b43b-aa19d97ea821`.
- `sshkey`: Ajoute la clé publique SSH donnée au fichier `allowed_keys` de l'utilisateur principal.

Remplacer l'exemple de clé ci-dessous par votre propre clé (elle se trouve généralement dans `~/.ssh/id_rsa.pub`)
- `console`: active la sortie du noyau et une invite de connexion sur un terminal donné. La valeur par défaut, `tty0`, correspond généralement à VGA. Peut être utilisé plusieurs fois, par exemple `console=tty0 console=ttyS0`
- `coreos.autologin`: Accédez directement à un shell sur une console donnée sans demander de mot de passe. Utile pour le dépannage mais utilisez avec prudence. Pour toute console qui ne reçoit normalement pas d'invite de connexion par défaut, assurez-vous de la combiner avec l'option console, par exemple. `console=tty0 console=ttyS0 coreos.autologin=tty1 coreos.autologin=ttyS0`. Sans aucun argument, il permet l'accès sur toutes les consoles. Notez que pour la console VGA, les invites de connexion se trouvent sur des terminaux virtuels (`tty1`, `tty2`, etc.) et non sur la console VGA elle-même (`tty0`).
- `coreos.first_boot=1`: Télécharge une configuration Ignition et l'utilise pour configurer le système démarré. Les configurations de boot sont générées à partir des configurations de conteneur Linux. Lorsqu'un système de fichiers local est utilisé pour la partition racine, il faut utiliser ce paramètre uniquement lors du premier démarrage.
- `coreos.config.url`: Télécharge la configuration Ignition à partir de l'URL spécifiée. Les schémas `http`, `https`, `s3` et `tftp` sont pris en charge.
- `ip`: Configure la mise en réseau statique temporaire pour `initramfs`. Ce paramètre n'influence pas la configuration réseau finale du nœud et est surtout utile pour le provisionnement au premier démarrage des systèmes dans des environnements sans DHCP.

Ceci est un exemple de fichier `pxelinux.cfg` qui suppose que `Container Linux` est la seule option. On doit pouvoir le copier dans `/var/lib/tftpboot/pxelinux.cfg/default` après avoir fourni une URL de configuration Ignition:

```
default coreos
prompt 1
timeout 15

display boot.msg

label coreos
  menu default
  kernel coreos_production_pxe.vmlinuz
  initrd coreos_production_pxe_image.cpio.gz
  append coreos.first_boot=1
coreos.config.url=https://example.com/pxe-config.ign
```

Voici un exemple de configuration commun qui doit être situé à l'URL ci-dessus:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

systemd:
  units:
    - name: etcd2.service
```

```
      enable: true

passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGdByTgSVHq...
```

Télécharger le noyau et initramfs

Dans la configuration ci-dessus, on peut voir qu'une image du noyau et un fichier initramfs sont nécessaires. Télécharger ces deux fichiers dans la racine `tftp`.

Les fichiers `coreos_production_pxe.vmlinuz.sig` et `coreos_production_pxe_image.cpio.gz.sig` peuvent être utilisés pour vérifier les fichiers téléchargés.

```
cd /var/lib/tftpboot
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_pxe.v
mlinuz
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_pxe.v
mlinuz.sig
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_pxe_i
mage.cpio.gz
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_pxe_i
mage.cpio.gz.sig
gpg --verify coreos_production_pxe.vmlinuz.sig
gpg --verify coreos_production_pxe_image.cpio.gz.sig
```

Démarrer la machine

Après avoir configuré le serveur PXE comme indiqué ci-dessus, on peut démarrer la machine cible en mode de démarrage PXE. La machine doit extraire l'image du serveur et démarrer dans Container Linux.

```
This is localhost.unknown_domain (Linux x86_64 3.10.10+) 19:53:36
SSH host key: 24:2e:f1:3f:5f:9c:63:e5:8c:17:47:32:f4:09:5d:78 (RSA)
SSH host key: ed:84:4d:05:e3:7d:e3:d0:b9:58:90:58:3b:99:3a:4c (DSA)
ens0: 10.0.2.15 fe80::5054:ff:fe12:3456
localhost login:
```

Se connecter

L'adresse IP de la machine doit être affichée sur le terminal pour plus de commodité. S'il n'apparaît pas immédiatement, appuyez plusieurs fois sur enter et il devrait s'afficher. Maintenant, on peut simplement utiliser SSH en utilisant l'authentification par clé publique:

```
ssh core@10.0.2.15
```

Installation sur un disque

Une fois démarré, il est possible d'installer `Container Linux` sur un disque local ou d'utiliser simplement le stockage local pour le système de fichiers racine tout en continuant d'amorcer `Container Linux` lui-même via PXE.

L'orqu'on prévoit d'utiliser Docker, il est recommandé d'utiliser un système de fichiers `ext4` local. Toutefois, `btrfs` est également disponible si nécessaire.

Installation avec le script

Il existe un programme d'installation simple qui va tout détruire sur le disque cible donné et installer Container Linux. Essentiellement, il télécharge une image, la vérifie avec gpg, puis la copie bit par bit sur disque. Une installation nécessite au moins 8 Go d'espace utilisable sur le périphérique.

Le script est autonome et se trouve sur GitHub et peut être exécuté à partir de n'importe quelle distribution Linux. On ne peut normalement pas installer `Container Linux` sur le même périphérique actuellement démarré. Cependant, l'ISO conteneur Linux ou tout liveCD Linux permettra à `Container Linux` de s'installer sur un périphérique non actif.

Lorsqu'on démarre `Container Linux` via PXE, le script d'installation est déjà installé. Par défaut, le script d'installation tente d'installer la même version et le même canal que ceux démarrés par PXE:

```
coreos-install -d /dev/sda -i ignition.json
```

Le fichier `ignition.json` doit inclure les informations utilisateur (en particulier une clé SSH) générées à partir d'une configuration Container Linux, sinon on ne pourra pas se connecter à la nouvelle instance Container Linux.

Pour configurer un système de fichiers racine `ext4` sur `/dev/sda`:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

storage:
```

```
  disks:
  - device: /dev/sda
    wipe_table: true
    partitions:
    - label: ROOT
  filesystems:
  - mount:
      device: /dev/disk/by-partlabel/ROOT
      format: ext4
      wipe_filesystem: true
      label: ROOT
```

Et ajouter `root=/dev/sda1` ou `root=LABEL=ROOT` aux options du noyau comme indiqué ci-dessus.

Pour configurer un système de fichiers racine `btrfs` sur `/dev/sda`:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

storage:
  disks:
  - device: /dev/sda
    wipe_table: true
    partitions:
    - label: ROOT
  filesystems:
  - mount:
      device: /dev/disk/by-partlabel/ROOT
      format: btrfs
      wipe_filesystem: true
      label: ROOT
```

Liste des options de coreos-install:

Option	Libellé
-d	DEVICE Installe `Container Linux` sur le périphérique donné.
-V	VERSION Version à installer (par exemple, la version actuelle)
-B	BOARD Conteneur Linux à utiliser
-C	CHANNEL Canal de diffusion à utiliser (par exemple, version bêta)
-o	OEM Type OEM à installer (par exemple, ami)
-c	CLOUD Insère une configuration cloud-init à exécuter au démarrage.
-i	IGNITION Insère une configuration d'allumage à exécuter au démarrage.
-b	BASEURL URL vers le miroir d’image (remplace BOARD)
-k	KEYFILE Remplace la clé GPG par défaut pour la vérification de la signature d'image
-f	IMAGE Installer le fichier image local non vérifié sur le disque au lieu de le récupérer
-n	Copier les unités réseau générées sur la partition racine.

Option	Libellé
-v	Super verbose, pour le débogage.

Installation sur RAID Logiciel

Container Linux prend en charge les unités de disques composites telles que les matrices RAID. Si le système de fichiers racine est placé sur un périphérique composite, il faut que `Container Linux` puisse trouver et monter le système de fichiers au début du processus de démarrage. Les entrées de partition GPT ont un GUID de type de partition qui spécifie le type de partition (par exemple, système de fichiers Linux); `Container Linux` utilise des GUID de type spécial pour indiquer qu'une partition est un composant d'un périphérique composite contenant le système de fichiers racine.

RAID permet de combiner plusieurs disques en un seul disque logique pour améliorer la fiabilité et les performances. Pour créer une grappe RAID logicielle lors de la mise en service d'un système Container Linux, utilisez la section storage.raid de `Container Linux` Config. Les composants RAID contenant le système de fichiers racine doivent avoir le type `GUID be9067b9-ea49-4f15-b4f6-f36f8c9e1818`. Toutes les autres matrices RAID ne doivent pas avoir ce GUID; le GUID de la partition RAID Linux `a19d880f-05fc-4d3b-a006-743f0f84911e` est recommandé.

Pour placer le système de fichiers racine sur une matrice RAID:

- Créer les partitions de composant utilisées dans la matrice RAID avec le type de GUID `be9067b9-ea49-4f15-b4f6-f36f8c9e1818`.
- Créer une matrice RAID à partir des partitions le composant.
- Créer un système de fichiers nommé ROOT sur la matrice RAID.
- Supprimer l'étiquette ROOT du système de fichiers racine d'origine.

Exemple de configuration de conteneur Linux

Ce conteneur Linux Config crée des partitions sur `/dev/vdb` et `/dev/vdc` qui remplissent chaque disque, crée une matrice RAID nommée root_array à partir de ces partitions et crée enfin le système de fichiers racine sur la matrice. Pour empêcher tout démarrage par inadvertance à partir du système de fichiers racine d'origine, `/dev/vda9` est reformaté avec un système de fichiers ext4 vierge étiqueté «non utilisé».

Attention: Ceci effacera `/dev/vdb` et `/dev/vdc`.

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

storage:
  disks:
    - device: /dev/vdb
      wipe_table: true
      partitions:
       - label: root1
         type_guid: be9067b9-ea49-4f15-b4f6-f36f8c9e1818
```

```
    - device: /dev/vdc
      wipe_table: true
      partitions:
       - label: root2
         type_guid: be9067b9-ea49-4f15-b4f6-f36f8c9e1818
  raid:
    - name: "root_array"
      level: "raid1"
      devices:
        - "/dev/vdb1"
        - "/dev/vdc1"
  filesystems:
    - name: "ROOT"
      mount:
        device: "/dev/md/root_array"
        format: "ext4"
        label: "ROOT"
    - name: "unused"
      mount:
        device: "/dev/vda9"
        format: "ext4"
        wipe_filesystem: true
        label: "unused"
```

Limitations

- Les autres partitions système, telles que `USR-A`, `USR-B`, `OEM` et `EFI-SYSTEM`, ne peuvent pas être placées sur une grappe RAID logicielle.
- Les composants RAID contenant le système de fichiers racine doivent être des partitions sur un périphérique partitionné GPT, et non des périphériques de disque entier ou des partitions sur un disque partitionné MBR.
- `/etc/mdadm.conf` ne peut pas être utilisé pour configurer un ensemble RAID contenant le système de fichiers racine.
- `Ignition` ne pouvant pas modifier le GUID de type des partitions existantes, la partition ROOT par défaut ne peut pas être réutilisée en tant que composant d'une matrice RAID.

Installation bare metal

Télécharger l'image de conteneur Linux brute à partir de CoreOS.

```
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_image
.bin.bz2
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_image
.bin.bz2.DIGESTS
wget
```

```
https://stable.release.core-os.net/amd64-usr/current/coreos_production_image
.bin.bz2.DIGESTS.asc
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_image
.bin.bz2.DIGESTS.sig
wget
https://stable.release.core-os.net/amd64-usr/current/coreos_production_image
.bin.bz2.sig
```

Une bonne pratique est de toujours vérifier l'image téléchargée.

```
gpg --verify coreos_production_image.bin.bz2.DIGESTS.sig
gpg --verify coreos_production_image.bin.bz2.sig
```

Créer un volume ZFS pour le lecteur de disque.

```
zfs create -V 32G -s -o reservation=none -o volmode=dev zroot/coreos
```

Décompresser l'image dans votre nouveau volume ZFS:

```
bzcat coreos_production_image.bin.bz2 > /dev/zvol/zroot/coreos
```

Prendre un Instantané ZFS

```
zfs snap zroot/coreos@vanilla
```

À ce stade, on peut démarrer le conteneur VM Linux, mais il ne sera pas accessible.

Configuration des accès

Par défaut, aucun mot de passe ni aucun autre moyen ne permet de se connecter à un nouveau système `Container Linux`. Le moyen le plus simple de configurer des comptes, d'ajouter des unités `systemd` et bien plus encore consiste à utiliser une configuration `Container Linux`.

Une configuration de conteneur Linux qui spécifie une clé SSH pour l'utilisateur principal mais n'utilise aucun autre paramètre ressemble à ceci:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

# Cette config est destinée à être consommée par le transpiler de config,
qui va
# générer la configuration d'allumage correspondante. Ne pas passer cette
```

```
config directement
# aux instances de Container Linux.


passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGdByTgSVHq.......
```

Les variables de substitution `{PRIVATE_IPV4}` et `{PUBLIC_IPV4}` référencées dans d'autres documents ne sont pas prises en charge sur libvirt.

Il existe plusieurs options pour accéder à la VM et la configurer:

- Ajouter la (les) clé (s) SSH à `~core/.ssh/registered_keys`
- Configurer l'accès via une configuration hébergée

Ajout de clé(s) ssh

Depuis un disque live, monter la partition racine

```
mount /dev/sda9 /media
```

Copier la clé publique sur la machine virtuelle et l'ajouter à `~core/.ssh/authorized_keys`

Redémarrer la VM sans le disque live et se reconnecter en ssh en utilisant l'authentification par clé publique.

```
ssh -l core <adresse ip>
```

Accès via une configuration hébergée

Cette section est réalisée à partir de n'importe quel hôte du même réseau que la machine virtuelle.

Un serveur Web est requis pour héberger un fichier de configuration. Par exemple avec nginx

Ecrire la configuration en `json` ou au format `yml`, puis la convertir en json en utilisant le transpiler coreos.

YML:

```
passwd:
```

```
  users:
    - name: core
      ssh_authorized_keys:
        - "ssh-rsa AAAAB3NzaC1...
```

JSON:

```
{
    "ignition": {
        "config": {},
        "security": {
            "tls": {}
        },
        "timeouts": {},
        "version": "2.2.0"
    },
    "networkd": {},
    "passwd": {
        "users": [
            {
                "name": "core",
                "sshAuthorizedKeys": [
                    "ssh-rsa AAAA..."
                ]
            }
        ]
    },
    "storage": {},
    "systemd": {}
}
```

Configurer les paramètres du noyau pour la configuration hébergée

A partir d'un livecd, ajouter des paramètres de noyau pour indiquer au conteneur Linux d'utiliser la configuration hébergée, dans un fichier `grub.cfg` de la partition OEM de la machine virtuelle Linux conteneur.

```
ls -l /dev/disk/by-partlabel

lrwxrwxrwx 1 root root 10 Aug 19 13:42 BIOS-BOOT -> ../../sda2
lrwxrwxrwx 1 root root 10 Aug 19 13:42 EFI-SYSTEM -> ../../sda1
lrwxrwxrwx 1 root root 10 Aug 19 13:42 OEM -> ../../sda6
lrwxrwxrwx 1 root root 10 Aug 19 13:43 OEM-CONFIG -> ../../sda7
lrwxrwxrwx 1 root root 10 Aug 19 13:42 ROOT -> ../../sda9
lrwxrwxrwx 1 root root 10 Aug 19 13:42 USR-A -> ../../sda3
lrwxrwxrwx 1 root root 10 Aug 19 13:42 USR-B -> ../../sda4
```

```
mount /dev/sda6 /media
```

```
/media/grub.cfg

linux_append="coreos.autologin coreos.first_boot=1
coreos.config.url=http://<IP Address>/config.json"
```

Redémarrer la machine virtuelle sans ISO Live Linux.

Configuration des Unités de stockage

Les configurations `Container Linux` peuvent être utilisées pour formater et attacher des systèmes de fichiers supplémentaires aux nœuds `Container Linux`, que ce stockage soit fourni par une plateforme cloud, un disque physique, un réseau SAN ou un système NAS sous-jacent. Ceci est fait en spécifiant comment les partitions doivent être montées dans la configuration, puis en utilisant une unité de montage `systemd` pour monter la partition.

Selon la convention de `Systemd`, les noms d'unité de montage dérivent du point de montage cible, les barres obliques internes étant remplacées par des tirets et l'extension .mount ajoutée. Une unité montée sur `/var/www` est donc nommée `var-www.mount`.

Les unités de montage nomment le système de fichiers source et le point de montage cible, ainsi que le type de système de fichiers. `Systemd` monte les systèmes de fichiers définis dans ces unités au démarrage.

Utiliser le stockage attaché pour Docker

Les conteneurs Docker peuvent être très volumineux et le débogage d'un processus de construction facilite l'accumulation de centaines de conteneurs. Il est avantageux d'utiliser le stockage attaché pour étendre votre capacité pour les images de conteneur.

L'exemple suivant formate un périphérique au format ext4 puis le monte sur `/var/lib/docker`, où Docker stocke les images:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

storage:
  filesystems:
    - name: ephemeral1
      mount:
```

```
        device: /dev/xvdb
        format: ext4
        wipe_filesystem: true
systemd:
  units:
    - name: var-lib-docker.mount
      enable: true
      contents: |
        [Unit]
        Description=Mount ephemeral to /var/lib/docker
        Before=local-fs.target
        [Mount]
        What=/dev/xvdb
        Where=/var/lib/docker
        Type=ext4
        [Install]
        WantedBy=local-fs.target
    - name: docker.service
      dropins:
        - name: 10-wait-docker.conf
          contents: |
            [Unit]
            After=var-lib-docker.mount
            Requires=var-lib-docker.mount
```

Création et montage d'un fichier de volume btrfs

Container Linux utilise `ext4` + `overlayfs` pour fournir un système de fichiers en couches à la partition racine. Si vous souhaitez utiliser `btrfs` pour les conteneurs Docker, on peut le faire avec deux unités `systemd`: une qui crée et formate un fichier de volume `btrfs` et une autre qui le monte.

L'exemple suivant formate un volume de 25 Go `btrfs` et le monte dans `/var/lib/docker`. On peut vérifier que Docker utilise le pilote de stockage `btrfs` une fois que le service Docker a démarré en exécutant les informations `sudo docker`. On recommande d'allouer au maximum 85% de l'espace disque disponible pour un système de fichiers `btrfs`, car `journald` aura également besoin d'espace sur le système de fichiers hôte.

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

systemd:
  units:
    - name: format-var-lib-docker.service
      contents: |
        [Unit]
        Before=docker.service var-lib-docker.mount
        RequiresMountsFor=/var/lib
```

```
        ConditionPathExists=!/var/lib/docker.btrfs
        [Service]
        Type=oneshot
        ExecStart=/usr/bin/truncate --size=25G /var/lib/docker.btrfs
        ExecStart=/usr/sbin/mkfs.btrfs /var/lib/docker.btrfs
    - name: var-lib-docker.mount
      enable: true
      contents: |
        [Unit]
        Before=docker.service
        After=format-var-lib-docker.service
        Requires=format-var-lib-docker.service
        [Mount]
        What=/var/lib/docker.btrfs
        Where=/var/lib/docker
        Type=btrfs
        Options=loop,discard
        [Install]
        RequiredBy=docker.service
```

Sans la déclaration de `ConditionPathExists=!/var/lib/docker.btrfs`, `systemd` reformaterait le système de fichiers `btrfs` à chaque démarrage de la machine.

Monter les exportations NFS

Cet extrait de la configuration de `Container Linux` monte une exportation NFS dans le répertoire `/var/www` du noeud Container Linux.

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

systemd:
  units:
    - name: var-www.mount
      enable: true
      contents: |
        [Unit]
        Before=remote-fs.target
        [Mount]
        What=nfs.example.com:/var/www
        Where=/var/www
        Type=nfs
        [Install]
```

```
        WantedBy=remote-fs.target
```

Dépendances de montage

Pour déclarer qu'un autre service dépend de ce montage, il faut nommer l'unité de montage dans les propriétés `After` et `Requires` de l'unité dépendante:

```
[Unit]
After=var-www.mount
Requires=var-www.mount
```

Si le montage échoue, les unités dépendantes ne démarreront pas.

Utilisation de Container Linux

Pour pouvoir se connecter à `Container linux` le client doit fournir une clé d'authentification.

Pour démarrer l'agent utilisateur sur le client, taper:

```
eval $(ssh-agent)
```

Se connecter à la machine conteneur Linux via SSH en tant que noyau utilisateur.

```
$ ssh core@[ip-delamachine
CoreOS (beta)
```

Avec `Vagrant`, la manière de se connecter est un peu différente:

```
$ ssh-add ~/.vagrant.d/insecure_private_key
Identity added: /Users/core/.vagrant.d/insecure_private_key
(/Users/core/.vagrant.d/insecure_private_key)
$ vagrant ssh core-01
CoreOS (beta)
```

Container Linux fournit trois outils essentiels:

- la gestion de conteneurs
- la découverte de services
- la gestion de processus

Docker: la gestion des conteneurs

Docker, est l'environnement d'exécution des applications et du code. Il est installé sur chaque machine Linux conteneur.Chacun des services (serveur Web, mise en cache, base de données) doivent être créés puis connectés ensemble en lisant et en écrivant sur etcd. On peut rapidement essayer un conteneur busybox minimal de deux manières différentes:

Exécuter une commande dans le conteneur puis l'arrêter:

```
docker run busybox /bin/echo hello world
```

Ouvrir une invite du shell à l'intérieur du conteneur:

```
docker run -i -t busybox /bin/sh
```

etcd: la découverte de services

etcd est un magasin de stockage clé-valeur en haute dispo (à l'aide du protocole Raft), distribué et résilient. Son but est de partager des données entre toutes les instances de CoreOS, pour peu qu'elles utilisent etcd. Écrire et récupérer des données dans etcd se fait en utilisant l'API REST sur un des clients.

etcd tourne sur l'hôte et non pas dans le conteneur.

etcd possède des fonctionnalité intéressantes, telles que :

- TTL sur des entrées ou des répertoires,
- attente qu'une entrée change et historique des changements,
- ordonnancement des clés afin de simuler une queue,
- répertoires cachés,
- statistiques sur le cluster etcd ou juste un noeud, et encore plus.

Avec un tel outils, il devient possible de diffuser une configuration sur tous les hôtes sans avoir besoin d'installer un autre logiciel.

```
$ curl -X PUT -L http://127.0.0.1:4001/v2/keys/hello -d value="world"
{"action":"set","node":{"key":"/hello","value":"world","modifiedIndex":9664,
"createdIndex":9664}}
```

```
$ curl -L http://127.0.0.1:4001/v2/keys/hello
{"action":"get","node":{"key":"/hello","value":"world","modifiedIndex":9513,
```

```
"createdIndex":9513}}
```

Mais on peut aussi utiliser `etcdctl`, une interface de ligne de commande pour `etcd` préinstallée sur `Container Linux`.

```
$ etcdctl set /hello you
you
```

```
core@core-<strong>02</strong> ~ $ etcdctl get /hello
you
```

Avec etcd les hôtes partagent des données et les conteneurs Docker qui tournent sur chacun des hôtes. Mais ces conteneurs sont un peu « collés » aux hôtes; CoreOS semble un OS éminemment versatile : on peux l'installer très vite et etcd permet de partager des données entre tous, alors pourquoi les conteneurs ne semblent pas aussi mobiles ?

fleet: la gestion de processus

fleet contrôle le démon `systemd` des instances CoreOS via `d-bus`, et est accessible par socket (Unix ou réseau). Il utilise `etcd` (celui qui écoute sur localhost, par défaut) pour communiquer avec les autres instances `fleet` sur les autres hôtes.

`Systemd` est une alternative au système de démarrage hérité de System V, mais il fait plus de choses. I remplace les scripts shell par des fichiers de configuration appelés « units ».

`Fleet` déploie les units (et les conteneurs Docker peuvent être gérés par des units) selon certaines stratégies :

- Déploiement des units sur tous les hôte qui font tourner fleet. Ce sont les global units.
- Déploiement d'une unit sur un hôte si une certaine unit n'y est pas. Utile pour la haute dispo.
- Ou au contraire, forcer deux units à tourner sur le même hôte.
- Déploiement d'une unit sur un hôte taggé : chaque démon `fleet` peut être taggé afin de le catégoriser : position géographique, fonction, … Ces tags peuvent être positionnés via `cloud-config`.

Quand on ne fait rien de particulier, lorsqu'on demande à `fleet` de faire tourner une unit, il le fait sur l'hôte sur lequel on est. Lorsq'on détruit (ou éteind) cet hôte, l'unit (le service, donc) sera lancé sur un autre hôte sans qu'on ai rien à faire quoi que ce soit.

Bien sûr, on peut voir sur quel hôte tourne l'unit.

Par exemple sur un cluster `fleet` composé de 3 machines :

```
$ fleetctl list-machines
MACHINE         IP              METADATA
1aaab8e5...     172.17.8.102    -
379ba5b7...     172.17.8.101    -
```

```
57d0cc63...     172.17.8.103    -
```

Créer un fichier unit:

```
$ cat m.service
[Unit]
Description=MyApp
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill busybox1
ExecStartPre=-/usr/bin/docker rm busybox1
ExecStartPre=/usr/bin/docker pull busybox
ExecStart=/usr/bin/docker run --name busybox1 busybox /bin/sh -c "while
true; do echo Hello World; sleep 1; done"
ExecStop=/usr/bin/docker stop busybox1
```

Lancer cet unit:

```
$ fleet start m.service
```

Allons voir où il tourne :

```
core@core-01 ~ $ fleetctl list-unit-files
UNIT            HASH    DSTATE          STATE           TARGET
m.service       391d247 launched        launched
379ba5b7.../172.17.8.101
```

172.17.8.101 est l'IP de l'hôte sur lequel on a lancé l'unit.

Tuer l'unit sur cet hôte, attendre un peu (quelques secondes) puis, sur un autre hôte :

```
$ fleetctl list-unit-files
UNIT HASH DSTATE STATE TARGET
m.service 391d247 launched launched 1aaab8e5.../172.17.8.102
```

Le service s'est déplacé !

Mais pas ses données. Pour ça, on peut utiliser `flocker`, qui fait de la migration de données d'environnements conteneurisés.

Gestion des mises à jour

CoreOS est conçu pour que utiliser des conteneurs et il s'occupe de l'hôte. Il y a exactement ce dont on a besoin, pas plus pas moins et la mise à jour est faite automatiquement et de manière transparente. Toutes les briques sont là pour ça : on s'occupe des conteneurs (leur vie, leur migration, leur mort), CoreOS gère le reste. On donne juste quelques indications : redémarre l'hôte quand la mise à jour est effectuée, redémarre les hôtes un par un, ne redémarre pas (fleet fait migrer less conteneurs automatiquement)

Mise à jour automatique

La mise à jour se fait en swappant entre une partition active et un partition passive. La mise à jour est installé sur la partition passive, au redémarrage elle devient active.

Stratégies de redémarrage sur les mises à jour

Il est important de noter que les mises à jour sont toujours téléchargées sur la partition passive lorsqu'elles sont disponibles. Un redémarrage est la dernière étape de la mise à jour, où les partitions actives et passives sont échangées (instructions de restauration). Ces stratégies contrôlent comment ce redémarrage se produit:

Désactiver le démon de mises à jour automatiques

Si on ne souhaitz pas installer les mises à jour sur la partition passive et éviter le processus de mise à jour en cas de redémarrage en cas d'échec, on peut désactiver le service `update-engine` manuellement à l'aide de la commande `sudo systemctl stop update-engine` (elle sera automatiquement activée au prochain redémarrage).

Pour désactiver les mises à jour automatiques de façon permanente, on peut le configurer dans la configuration Linux du conteneur. Cet exemple arrêtera `update-engine`, qui exécute les mises à jour, et `locksmithd`, qui coordonne les redémarrages dans le cluster:

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.

systemd:
  units:
    - name: update-engine.service
      mask: true
    - name: locksmithd.service
```

```
      mask: true
```

Mise à jour derrière un proxy

Un accès Internet public est requis pour contacter CoreUpdate et télécharger les nouvelles versions de Container Linux. Si l'accès direct n'est pas disponible, le service update-engine peut être configuré pour utiliser un proxy HTTP ou SOCKS à l'aide de variables d'environnement compatibles Curl, telles que HTTPS_PROXY ou ALL_PROXY. Voir la documentation de curl pour plus de détails.

```
# This config is meant to be consumed by the config transpiler, which will
# generate the corresponding Ignition config. Do not pass this config
directly
# to instances of Container Linux.
systemd:

  units:
    - name: update-engine.service
      dropins:
        - name: 50-proxy.conf
          contents: |
            [Service]
            Environment=ALL_PROXY=http://proxy.example.com:3128
```

Les variables d'environnement proxy peuvent également être définies à l'échelle du système.

Déclencher manuellement une mise à jour

On peut forcer une vérification de mise à jour, qui ignorera tous les paramètres de limitation de débit configurés dans `CoreUpdate`.

```
$ update_engine_client -check_for_update
[0123/220706:INFO:update_engine_client.cc(245)] Initiating update check and
install.
```