CoreOS: l'OS Container Linux

Table of Contents

- CoreOS: l'OS Container Linux
  - Ignition
    - Créer une configuration Ignition
    - Paramètres utilisateur
    - Paramètres du système de fichiers
    - Paramètres système
    - Installation de nouveaux packages
  - Fedora CoreOS
    - Installation automatisée
    - Références ostree
    - Rebasage
    - Analyse du système de fichiers
    - Gestion des mises à niveau du système
  - Flatcar Container Linux
    - Installation
    - Mises à jour automatique.
    - Configurations Linux du conteneur

CoreOS est un système d'exploitation léger open source basé sur le noyau Linux, concrètement **CoreOS** est un système Linux allégé au maximum. Un noyau et **systemd**, tout simplement.

À la suite de son acquisition de **CoreOS**, **Red Hat** a annoncé qu'il fusionnerait **CoreOS Container Linux** avec le projet **Atomic de Red Hat**, pour créer un nouveau système d'exploitation, **Red Hat CoreOS**, tout en alignant la communauté open source de Fedora Project autour de **Fedora CoreOS**, cependant il existe une alternative **Flatcar Container Linux**.

Ce système est open source. Les sources des différents composants sont disponibles sur github.

Cet article présente l'outil de provisionnement **Ignition**, **Fedora CoreOS** et **Flatcar Container Linux** (fork de **Red Hat CoreOS**).

Ignition

Ignition est un outil de provisionnement de bas niveau qui permet une configuration de démarrage précoce. La partie essentielle est un fichier de configuration qui est lu lors du processus de démarrage initial. Ce fichier **.ign** n'est pas écrit directement, un fichier est écrit en syntaxe yaml, puis un

transpileur[1] est utilisé afin d'adapter le fichier au format de fichier **Ignition**.

Créer une configuration Ignition

Les configurations **Ignition** sont des fichiers JSON standard qui ne sont pas encodés dans un joli format. Ils peuvent être longs et difficiles à lire ou à modifier. FCOS propose un format compatible appelé Fedora CoreOS Configuration (FCC), un fichier de configuration au format YAML plus facile à lire/écrire. Pour générer des fichiers **Ignition** à partir d'un FCC, on peut utiliser l'outil Fedora CoreOS Configuration Transpiler[2].

Cet outil est facile à utiliser, mais il faut d'abord créer un fichier FCC. Voici un exemple simple (example-fcc.yaml) qui définit une clé SSH publique pour le noyau utilisateur, l'utilisateur cloud par défaut dans FCOS :

```
variant: fcos
version: 1.0.0
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-rsa AAAAB3NzaC1yc...
```

Après avoir écrit le fichier FCC, on doit le traduire en un fichier Ignition. Télécharger la dernière version de **fcct** et l'installer localement (/usr/local/bin est le meilleur choix pour les binaires compilés ou fournis par l'utilisateur).

Maintenant, exécuter la commande suivante pour transformer le fichier FCC en un fichier de configuration Ignition :

```
fcct -input example-fcc.yaml -output example-ignition.ign
```

Lorsqu'on exécute un système Fedora/CentOS/Red Hat Enterprise Linux et que **SELinux** est activé (comme il se doit), **SELinux** bloquera la création de l'instance puisque le processus qemu-kvm essaie d'accéder aux fichiers dans un répertoire sans le contexte **virt_image_t**. Pour résoudre ce problème, on a deux options : mettre **SELinux** en mode permissif ou renommer le répertoire contenant les fichiers Ignition.

Pour activer le mode permissif :

```
sudo setenforce 0
sed -i '/^SELINUX=/ s/enforcing/permissive/g' /etc/selinux/config
```

Alternativement, pour changer le contexte du fichier :

```
sudo semanage fcontext -a -t virt_image_t
'/path/to/ignitions(/.*)?'
sudo restorecon -Rvv /path/to/ignitions
```

Les deux options sont interchangeables.

On peut utiliser les configurations **Ignition** pour gérer le stockage et les utilisateurs ; créer des fichiers, des répertoires et des unités systemd ; et injecter des clés ssh.

Après avoir personnalisé le fichier FCC, il faut générer le fichier **Ignition** à l'aide de l'outil **fcct**:

```
fcct -input example-fcc-systemd.yaml -output example-ignition-systemd.json
```

FCC autorise essentiellement trois types d'options de configuration :

- Utilisateurs et groupes
- Système de fichiers
- Unités système

Paramètres utilisateur

La configuration **Ignition** minimale, ainsi que certaines métadonnées requises, doivent disposer d'informations sur les utilisateurs initiaux du système autorisés à se connecter au système. Fedora CoreOS fournit un noyau utilisateur initial qui peut être utilisé pour administrer le système. Si on souhaite utiliser cet utilisateur, la configuration minimale doit au moins attribuer un mot de passe de connexion correspondant ou une clé ssh.

Un exemple pourrait ressembler à ceci :

```
variant: fcos
version: 1.0.0
passwd:
  users:
  - name: core
    ssh_authorized_keys:
    - "ssh-rsa AAAAB3Nza...."
  - name: matthias
    ssh_authorized_keys:
    - "ssh-rsa AAAAB4adsa...."
    password_hash: "$6$rounds=4096$yspKBgKn..."
    groups:
    - wheel
```

Ici, l'utilisateur central du système dispose d'une clé ssh publique fournie dans la section ssh*authorized*keys et est donc autorisé à se connecter au système via ssh. L'utilisateur créé en plus matthias a un **password_hash configuré**, donc une connexion régulière basée sur un mot de passe est également possible.

Évidemment, autoriser une connexion régulière basée sur un mot de passe nécessite une méthode de hachage sûre (ne pas choisir MD-5 !). Cela peut être fait avec les outils Linux habituels tels que :

```
mkpasswd --method=SHA-512 --rounds=4096
```

Le mot de passe haché peut ensuite être utilisé pour la configuration FCC. L'exemple de configuration affecte également l'utilisateur matthias au groupe wheel pour autoriser les opérations privilégiées.

Outre ces paramètres standard, des options plus sophistiquées telles que la configuration de comptes système sans répertoire personnel ou la configuration du shell de connexion utilisé sont également disponibles.

Paramètres du système de fichiers

Ignition permet de mettre en place le stockage lors de la phase de configuration. Cela inclut le partitionnement, le formatage et le montage de périphériques de disque supplémentaires et des systèmes de fichiers correspondants. Pour ce faire, il faut d'abord créer une liste de périphériques de disque. Les chemins absolus vers les périphériques doivent généralement être fournis à l'aide des liens symboliques `/dev/disk/by-…`, qui sont cohérents et non susceptibles de changer. Une liste de partitions pour chaque périphérique peut ensuite être définie, avec d'autres options pour effacer les tables de partitions et les partitions préexistantes.

La création de systèmes de fichiers sur les périphériques de disque configurés se fait ensuite facilement en les listant dans la section des systèmes de fichiers correspondante. Les paramètres incluent ici le chemin du périphérique, le système de fichiers souhaité et d'autres paramètres tels que les étiquettes.

Il convient de mentionner que, bien qu'**Ignition** puisse gérer divers systèmes de fichiers tels que ext4 et xfs, des options plus sophistiquées telles que la configuration de LVM ne sont pas possibles.

Un exemple minimal de configuration du stockage pourrait alors ressembler à ceci :

```
variant: fcos
version: 1.0.0
storage:
  disks:
  - device: <path to disk device>
    wipe_table: true
    partitions:
    - label: <unique partition label>
      number: 0
      wipe_partition_entry: true
  filesystems:
  - path: <mount path of device during configuration>
    device: <path to disk device>
    wipe_filesystem: true
    format: ext4
    label: <unique file system label>
```

Ignition n'offre aucune possibilité d'exécuter des commandes shell pendant le processus de configuration (par exemple la manière habituelle de définir le nom d'hôte à l'aide de **hostnamectl** sur les systèmes contrôlés par systemd ne fonctionne pas immédiatement). Il existe deux solutions

afin de définir le contenu d'un fichier:

- Configurer un service systemd oneshot pour exécuter une commande
- Écrire directement dans le fichier de configuration correspondant

Dans l'exemple suivant, on opte pour la deuxième solution, en utilisant la section de stockage du fichier de configuration :

```
variant: fcos
version: 1.0.0
storage:
  files:
  - path: /etc/hostname
    overwrite: true
    contents:
      inline: <your-hostname>
```

Paramètres système

La section **systemd** de la configuration FCC permet la configuration des unités systemd. Il est possible de créer de nouvelles unités ainsi que de modifier celles existantes avec par ex. fichiers de configuration intégrés.

Pour revenir à l'exemple précédent, la création d'un nouveau service oneshot pour configurer le nom d'hôte lors du premier démarrage peut ressembler à ceci :

```
variant: fcos
version: 1.0.0
systemd:
  units:
  - name: set-hostname.service
    enabled: true
    contents: |
      [Unit]
      Description=Set the hostname

      [Service]
      Type=oneshot
      ExecStart=/usr/bin/hostnamectl set-hostname your-hostname

      [Install]
      WantedBy=multi-user.target
```

Grâce à la nature sans démon de **Podman**, on peut exécuter, démarrer ou arrêter les conteneurs dans une unité systemd et gérer facilement leur ordre de démarrage:

```
variant: fcos
```

```
version: 1.0.0
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-rsa AAAAB3Nza...

systemd:
  units:
    - name: hello.service
      enabled: true
      contents: |
        [Unit]
        Description=A hello world unit!
        After=network-online.target
        Wants=network-online.target
        [Service]
        Type=forking
        KillMode=none
        Restart=on-failure
        RemainAfterExit=yes
        ExecStartPre=podman pull quay.io/gbsalinetti/hello-server
        ExecStart=podman run -d --name hello-server -p 8080:8080
quay.io/gbsalinetti/hello-server
        ExecStop=podman stop -t 10 hello-server
        ExecStopPost=podman rm hello-server
        [Install]
        WantedBy=multi-user.target
```

Dans cet exemple, on a ajoute un simple fichier d'unité pour lancer une image qui exécute un serveur Web Go minimal et éphémère, puis imprime un message “Hello World”. Examiner la section systemd du fichier et sa sous-section units, qui contient une liste d'éléments représentant un ou plusieurs fichiers unitaires. Dans ce bloc, on peut définir autant d'unités que nécessaire, et elles seront créées et lancées au démarrage.

Installation de nouveaux packages

Avant de chercher à installer un package dans FCOS, il faut prendre en considération ceci: il n'y a pas d'outil **dnf** ou **yum** installé dans FCOS, et **rpm-ostree** (construit au-dessus de la bibliothèque **libostree**) est le gestionnaire de paquets par défaut. Les modifications sont validées en interne et les systèmes sont redémarrés pour appliquer la nouvelle couche.

La commande `rpm-ostree status --json` peut être utilisée pour imprimer des états étendus.

Pour installer un package on doit utiliser la commande `rpm-ostree install` :

```
$ sudo rpm-ostree install buildah
```

Le nouveau commit a ajouté une nouvelle couche et la commande `rpm-ostree status` affiche le package Buildah qu'on a installé auparavant. on peut superposer autant de packages que l'on souhaite ; par exemple, tcpdump :

```
sudo rpm-ostree install tcpdump
```

L'unité sistemd suivante permet d'automatiser une installation:

```
systemd:
  units:
    - name: overlay-install.service
      enabled: true
      contents: |
        [Unit]
        Description=Install Overlay Packages
        ConditionFirstBoot=yes
        Wants=network-online.target
        After=network-online.target
        After=multi-user.target
        Before=boot-complete.target
        [Service]
        Type=oneshot
        ExecStart=rpm-ostree install nano git docker-compose dnf htop
        # An alternative to --reboot
        ExecStartPost=rpm-ostree ex livefs --i-like-danger
        [Install]
        RequiredBy=boot-complete.target
        WantedBy=multi-user.target
```

Fedora CoreOS

Fedora CoreOS (**FCOS**), contrairement aux systèmes d'exploitation classiques, n'a pas de configuration au moment de l'installation. Au lieu de cela, une image disque générique est fournie, qui peut être configurée lors du processus de démarrage initial. **FCOS** prend en charge les déploiements sur des machines bare metal, dans un environnement cloud (par exemple en utilisant des services comme Amazon EC2) et sur des machines virtuelles locales, avec les possibilités correspondantes de fournir les paramètres de configuration initiaux.

Une liste détaillée des paquets dans l'image est disponible ici

Par défaut on trouvera les paquets suivants:

- **ignition** ext un utilitaire de configuration système de bas niveau qui est exécuté lors du démarrage dans l'initramfs de la machine.
- **moby-engine** est version sans marque de Docker-CE.
- **rpm-ostree**

- **podman**[3] est le runtime de conteneur par défaut, **Skopeo** et **Docker**[4] sont également installés). **Podman** est sans démon et relégue **Docker** uniquement dans les scénarios où la communication avec son socket Unix est obligatoire.

La bibliothèque **libostree** propose une API pour manipuler les transactions atomiques sur systèmes de fichiers immuables en les gérant comme des arbres entiers. La commande **ostree** est l'outil par défaut utilisé pour gérer ces modifications. Il existe de nombreuses liaisons de langage utiles pour créer nos propres implémentations ; par exemple, **ostree-go** est la liaison de langage pour Golang.

Installation automatisée

Il existe deux façons d'automatiser l'installation avec Ignition :

- Exécuter une installation via PXE
- Exécuter une installation à l'aide de Live ISO avec une configuration **Ignition** intégrée

Exécution de l'installation via PXE

Pour l'installation PXE, on peut utiliser Libvirt/iPXE pour tester facilement une installation. Après avoir copié les images live kernel/initramfs localement, créer la configuration ipxe suivante :

```
#!ipxe
set base-url http://192.168.122.1:8000
kernel ${base-url}/fedora-coreos-31.20200310.3.0-live-kernel-x86_64 ip=dhcp
rd.neednet=1 console=tty0 ignition.firstboot ignition.platform.id=metal
ignition.config.url=https://dustymabe.com/2020-04-04/automated_install.ign
initrd ${base-url}/fedora-coreos-31.20200310.3.0-live-initramfs.x86_64.img
boot
```

Ensuite, on peut lancer une VM et regarder l'installation automatisée :

```
virt-install --name pxe --network bridge=virbr0 --memory 4096 --disk size=20
--pxe
```

On exécute libvirt en mode session (non privilégié). virbr0 est le nom du pont qui fait partie du réseau par défaut de libvirt qui est configuré avec <bootp file='http://192.168.122.1:8000/boot.ipxe'

.

Après un certain temps, on doit voir le script s'exécuter pour effectuer l'installation.

Le système redémarre ensuite et exécute la deuxième configuration de provisionnement **Ignition** lors du premier démarrage du système installé. À la suite de l'application de cette configuration, la console sur tty1 (console VGA) doit être connectée par défaut.

Installation via une configuration Ignition dans l'ISO

L'outil **coreos-installer** prend en charge l'intégration d'une configuration **Ignition** fournie dans l'image ISO en direct afin qu'un utilisateur puisse obtenir un flux de travail automatisé sans avoir à intercepter les invites grub/isolinux afin de spécifier un ignition.config.url sur la ligne de commande du noyau.

On peut également utiliser cette fonctionnalité pour automatiser une installation à l'aide de l'image ISO en direct. Après avoir téléchargé l'ISO (dans ce cas, fedora-coreos-31.20200310.3.0-live.x86_64.iso), on peut intégrer la configuration **Ignition** comme suit :

```
coreos-installer iso embed --config automatic_install.ign ./fedora-coreos-31.20200310.3.0-live.x86_64.iso
```

Maintenant, lorsqu'on démarrera l'ISO, il appliquera la configuration **Ignition** qui exécutera l'installation :

```
$ virt-install --name cdrom --network bridge=virbr0 --memory 4096 --disk size=20 --cdrom ./fedora-coreos-31.20200310.3.0-live.x86_64.iso
```

L'installation se déroulera exactement de la même manière que dans le cas Live PXE ci-dessus et l'utilisateur sera finalement connecté sur la console VGA de la machine.

Références ostree

On peut voir le delta entre les couches du système de fichiers avec la commande **ostree refs**, pour localiser les différentes références des couches validées :

```
[core@localhost ~]$ refs ostree
rpmostree/pkg/buildah/1.12.0-2.fc31.x86__64
rpmostree/base/0
ostree/0/1/1
rpmostree/pkg/tcpdump/14_3A4.9.3-1.fc31.x86__64
fedora : fedora/x86_64/coreos/stable
ostree/0/1/0
```

Les références peuvent être considérées comme des branches différentes, comme dans un référentiel Git. Les branches créées par **rpm-ostree**, commencent par le motif `rpmostree/pkg/`.

Pour inspecter les changements qui se sont produits dans une branche, on peut utiliser la commande **ostree ls**.

Rebasage

Le rebasage, permet de passer à une branche différente et modifier radicalement l'arborescence du

système de fichiers.

La commande **rpm-ostree rebase** peut prendre une branche comme argument et déplacer tout le système vers cette branche. Si on rappelle la sortie de la commande **ostree refs**, il y avait la ligne suivante : `fedora:fedora/x86_64/coreos/stable`. Cette sortie signifie qu'on est sur la branche stable de Fedora CoreOS. Si on veut basculer vers la branche testing, il suffit de rebaser, comme dans l'exemple suivant.

```
sudo rpm-ostree rebase -b fedora:fedora/x86_64/coreos/testing
```

La liste des packages est mise à jour pour passer de la branche stable à la branche testing. Maintenant, la sortie **ostree refs** est légèrement différente, la branche testing remplaçant stable :

```
[core@localhost ~]$ refs ostree
rpmostree/pkg/buildah/1.12.0-2.fc31.x86__64
rpmostree/base/0
ostree/0/1/1
rpmostree/pkg/tcpdump/14_3A4.9.3-1.fc31.x86__64
fedora:fedora/x86_64/coreos/testing
ostree/0/1/0
```

Analyse du système de fichiers

Certains répertoires ne sont pas accessibles en écriture ; par exemple, lorsqu'on essaye d'écrire dans `/usr`, on obtiens une erreur de système de fichiers en lecture seule. Lorsque le système démarre, seules certaines parties (comme `/var`) sont accessibles en écriture.

La conception de l'architecture ostree stipule que le contenu en lecture seule du système est conservé dans le répertoire `/usr`. Le répertoire `/var` est partagé par tous les déploiements de système et est accessible en écriture par les processus, et il existe un répertoire `/etc` pour chaque déploiement de système. Lors de modifications ou de mises à niveau du système, les modifications précédentes dans le répertoire `/etc` sont fusionnées avec la copie dans le nouveau déploiement.

Toutes ces modifications sont stockées dans un référentiel OSTree dans `/ostree/repo`, qui est un lien symbolique vers `/sysroot/ostree/repo`. Ce répertoire stocke tous les déploiements du système. On peut assimiler `/sysroot/ostree/repo` comme le `.gitdirectory` dans un référentiel **Git**.

Pour vérifier l'état actuel et identifier l'UUID de déploiement actif, utiliser la commande suivante :

```
[core@localhost ~]$ statut d'administrateur sudo ostree
* fedora-coreos
4ea6beed22d0adc4599452de85820f6e157ac1750e688d062bfedc765b193505.0
    Édition : 31.20200210.3.0
    spécification de référence d'origine : fedora:
fedora/x86_64/coreos/stable
```

Si on inspecte le dossier `/ostree` dans un système fraîchement installé, on trouve une

correspondance exacte qui contient notre arborescence de système de fichiers démarré :

```
[core@localhost ~]$ ls -al /ostree/boot.1/fedora-
coreos/19190477fad0e60d605a623b86e06bb92aa318b6b79f78696b06f68f262ad5d6/0/
total 8
drwxr-xr-x. 12 racine racine 253 24 février 16:52.
drwxrwxr-x. 3 racine racine 161 24 février 16:52 ..
lrwxrwxrwx. 2 racine racine 7 février 24 16:51 bin -> usr/bin
drwxr-xr-x. 2 racine racine 6 1er janvier 1970 démarrage
drwxr-xr-x. 2 racine racine 6 1er janvier 1970 dev
drwxr-xr-x. 77 racine racine 4096 11 mars 08:54 etc.
lrwxrwxrwx. 2 root root 8 février 24 16:51 home -> var/home
lrwxrwxrwx. 2 root root 7 24 février 16:51 lib -> usr/lib
lrwxrwxrwx. 2 racine racine 9 février 24 16:51 lib64 -> usr/lib64
lrwxrwxrwx. 2 root root 9 février 24 16:51 media -> run/media
lrwxrwxrwx. 2 racine racine 7 24 février 16:51 mnt -> var/mnt
lrwxrwxrwx. 2 racine racine 7 février 24 16:51 opt -> var/opt
lrwxrwxrwx. 2 racine racine 14 février 24 16:51 ostree -> sysroot/ostree
drwxr-xr-x. 2 racine racine 6 1er janvier 1970 proc
lrwxrwxrwx. 2 racine racine 12 février 24 16:51 racine -> var/roothome
drwxr-xr-x. 2 root root 6 Jan 1 1970 run
lrwxrwxrwx. 2 racine racine 8 février 24 16:51 sbin -> usr/sbin
lrwxrwxrwx. 2 racine racine 7 24 février 16:51 srv -> var/srv
drwxr-xr-x. 2 root root 6 Jan 1 1970 sys
drwxr-xr-x. 2 racine racine 6 1er janvier 1970 sysroot
drwxrwxrwt. 2 racine racine 6 mars 11 08:37 tmp
drwxr-xr-x. 12 racine racine 155 1er janvier 1970 usr
drwxr-xr-x. 4 racine racine 28 Mar 11 08:37 var
```

Le répertoire ci-dessus représente le déploiement démarré et est en fait un lien symbolique vers une référence sous `/ostree/deploy/fedora-cores/deploy/`. Le nom du répertoire correspondra à l'UUID imprimé par la sortie de la commande **ostree admin status**.

Au démarrage du système, certains répertoires de la racine du système de fichiers sont créés en tant que liens physiques vers l'arborescence de déploiement active. Étant donné que les liens physiques pointent vers le même numéro d'inode dans le système de fichiers, on peut vérifier le numéro d'inode du dossier `/usr` et celui du déploiement démarré :

```
[core@localhost ~]$ ls -alid /usr
3218784 drwxr-xr-x. 12 racine racine 155 1er janvier 1970 /usr
[core@localhost ~]$ ls -alid /sysroot/ostree/boot.1/fedora-
coreos/19190477fad0e60d605a623b86e06bb92aa318b6b79f78696b06f68f262ad5d6/0/us
r/
3218784 drwxr-xr-x. 12 racine racine 155 1er janvier 1970
/sysroot/ostree/boot.1/fedora-
coreos/19190477fad0e60d605a623b86e06bb92aa318b6b79f78696b06f68f262ad5d6/0/us
r/
```

Comme prévu, le numéro d'inode 3218784 est le même pour les deux répertoires, démontrant que le

contenu sous la racine du système de fichiers est composé du déploiement actif.

Que s'est-il passé lorsque on a installé le package Buildah avec rpm-ostree ? Après le redémarrage, un nouveau déploiement apparaît :

```
[core@localhost ~]$ statut d'administrateur sudo ostree
* fedora-coreos
ee678bde3c15d8cae34515e84e2b4432ba3d8c9619ca92c319b576a13029481d.0
Édition : 31.20200210.3.0
origine : <type d'origine inconnu>
fedora-coreos
4ea6beed22d0adc4599452de85820f6e157ac1750e688d062bfedc765b193505.0
(rollback)
Édition : 31.20200210.3.0
spécification de référence d'origine : fedora: fedora/x86_64/coreos/stable
```

On note l'étoile à côté du déploiement actif actuel. on s'attend à le trouver sous `/ostree/deploy/fedora-cores/deploy` avec l'ancien :

```
[core@localhost ~]$ ls -al /ostree/deploy/fedora-coreos/deploy/
total 12
drwxrwxr-x. 4 racine racine 4096 11 mars 10:18.
drwxrwxr-x. 4 racine racine 31 février 24 16:52 ..
drwxr-xr-x. 12 racine racine 253 24 février 16:52
4ea6beed22d0adc4599452de85820f6e157ac1750e688d062bfedc765b193505.0
-rw-r--r--. 1 racine racine 52 24 février 16:52
4ea6beed22d0adc4599452de85820f6e157ac1750e688d062bfedc765b193505.0.origin
drwxr-xr-x. 12 racine racine 253 11 mars 10:01
ee678bde3c15d8cae34515e84e2b4432ba3d8c9619ca92c319b576a13029481d.0
-rw-r--r--. 1 racine racine 87 11 mars 10:18
ee678bde3c15d8cae34515e84e2b4432ba3d8c9619ca92c319b576a13029481d.0.origin
```

Le répertoire `/ostree/deploy/fedora-cores` est aussi appelé **stateroot** ou **osname**, et il contient le deployment et tous ses engagements connexes. Dans chaque répertoire **stateroot**, il n'y a qu'un seul répertoire `var` qui est monté sous `/var` à l'aide d'une unité de montage systemd (représentée par le fichier `/run/systemd/generator/var.mount`).

Dans le dossier `/ostree/repo` on peut trouver les similitudes les plus proches avec **Git**:

```
[repo core@localhost]$ ls -al /ostree/repo/
total 16
drwxrwxr-x. 7 racine racine 102 11 mars 10:18.
drwxrwxr-x. 5 racine racine 62 Mar 11 10:18 ..
-rw-r--r--. 1 racine racine 73 24 février 16:52 config
drwxr-xr-x. 3 racine racine 23 Mar 11 09:59 extensions
-rw-------. 1 racine racine 0 24 février 16:51 .lock
drwxr-xr-x. 258 racine racine 8192 24 février 16:52 objets
drwxr-xr-x. 5 racine racine 49 24 février 16:51 refs
drwxr-xr-x. 2 racine racine 6 février 24 16:52 état
```

```
drwxr-xr-x. 3 racine racine 19 mars 11 10:18 tmp
```

On note les dossiers `refs` et `objects`, qui stockent respectivement les informations de branche et les objets versionnés. Ici, on a une correspondance exacte dans le dossier `.git` d'un référentiel **Git** typique.

Gestion des mises à niveau du système

rpm-ostree fonctionne au-dessus de la bibliothèque **ostree** et fournit une gestion des packages avec une approche atomique. Les mises à niveau du système sont également atomiques. La mise à niveau d'un système consiste simplement à superposer un nouveau commit au-dessus du système de fichiers existant, et on peut le faire facilement avec la commande **rpm-ostree upgrade** :

```
[core@localhost ~]$ sudo rpm-ostree upgrade -r
```

L'indicateur -r indique à rpm-ostree de redémarrer automatiquement une fois la mise à niveau terminée.

Bien qu'il permette aux utilisateurs de mettre à niveau les systèmes manuellement, FCOS fournit un service dédié qui gère les mises à niveau du système appelé **Zincati**. **Zincati** est un agent qui effectue des contrôles de mise à niveau périodiques et les applique.

Le comportement de **Zincati** peut être personnalisé. Les configurations par défaut sont déjà installées sous /usr/lib/zincati/config.d/, tandis que les utilisateurs peuvent appliquer des configurations personnalisées dans `/etc/zincati/configs.d/` et remplacer les valeurs par défaut.

Flatcar Container Linux

CoreOS Container Linux de **Red Hat**, le système d'exploitation axé sur les conteneurs, ayant atteint sa fin de vie, il existe une alternative **Flatcar Container Linux**, un “fork” destiné à le remplacer afin qu'il ne diffère pas “de manière significative” de la version de Red Hat sur le site Web https://www.flatcar.org/.

Installation

Avant d'utiliser le script **flatcar-install** il faut s'assurer que les paquets suivants sont installés :

- bash
- lbzip2 or bzip2
    - mount, lsblk (souvent trouvé dans le paquet util-linux)
    - wget

- grep
- cp, dd, mkfifo, mkdir, rm, tee (souvent trouvés dans le paquet GNU coreutils ou dans le cadre de busybox)
- udevadm (trouvé dans le paquet systemd-udev, ou pour les images Alpine dans eudev)
- gpg, gpg2 (trouvé dans gnupg2)
- gawk (souvent trouvé dans le paquet GNU gawk)

Le programme d'installation détruira tout sur le disque cible donné et installera **Flatcar Container Linux**. Essentiellement, il télécharge une image, la vérifie avec gpg, puis la copie bit par bit sur le disque. Une installation nécessite au moins 8 Go d'espace utilisable sur l'appareil.

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

Si on démarre **Flatcar Container Linux** via PXE, le script d'installation est déjà installé. Par défaut, le script d'installation tentera d'installer la même version et le même canal que ceux qui ont été démarrés par PXE

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

Au minimum `ignition.json` doit inclure des informations utilisateur (en particulier une clé SSH) générées à partir d'une configuration **Container Linux**, sinon on ne pourra pas se connecter à l'instance **Flatcar Container Linux**.

Lorsu'on installe sur VMware, passer **-o vmware_raw** pour installer l'image spécifique à VMware :

```
flatcar-install -d /dev/sda -i ignition.json -o vmware_raw
```

Le canal stable doit être utilisé par les clusters de production. Les versions de **Flatcar Container Linux** sont testées dans les canaux beta et alpha avant d'être promues.

Pour s'assurer d'installer la dernière version stable, utiliser l'option **-C**:

```
flatcar-install -d /dev/sda -C stable
```

Pour référence, voici le reste des options d'installation de flatcar :

Commande	**Description**
-d DEVICE	Installe **Flatcar Container Linux** sur le périphérique donné.

Commande	Description
-s EXPERIMENTAL	installe **Flatcar Container Linux** sur le plus petit disque non monté trouvé (taille minimale 10 Go). Il est recommandé de l'utiliser avec **-e** ou **-l** pour filtrer bloquer les périphériques par leurs numéros majeurs. Par exemple, **-e 7** pour exclure les périphériques de boucle ou **-l 8 259** pour certains types de disque (voir https://www.kernel.org/doc/Documentation/admin-guide/devices.txt.
-V VERSION	Version à installer (par exemple actuelle)
-B BOARD	carte à utiliser
-C CHANNEL	Canal de version à utiliser (par exemple, beta)
-l/e <M,..> EXPERIMENTAL	(utilisé avec -s) liste des principaux numéros de périphériques à intégrer/exclure lors de la recherche du plus petit disque.
-o OEM	Type OEM à installer (par exemple ami), en utilisant **flatcar_production_<OEM>_image.bin.bz2**
-c CLOUD	Insère une configuration cloud-init à exécuter au démarrage.
-i IGNITION	Insère une configuration Ignition à exécuter au démarrage.
-b BASEURL	URL vers le miroir d'image (remplace BOARD et CHANNEL)
-k KEYFILE	Remplace la clé GPG par défaut pour vérifier la signature de l'image
-f IMAGE	Installe le fichier image local non vérifié sur le disque au lieu de le récupérer
-n	Copie les unités réseau générées sur la partition racine.
-v	Mode super verbeux, pour le débogage.

Mises à jour automatique.

Flatcar Container Linux est conçu pour être mis à jour automatiquement avec différents horaires par canal (stable, beta, alpha, edge).

Pour désactiver temporairement les redémarrages de mise à jour, exécuter `sudo systemctl stop update-engine locksmithd` et, une fois terminé, `sudo systemctl start update-engine locksmithd`.

Pour désactiver définitivement les mises à jour automatiques, il n'est pas recommandé de masquer les services car cela rend plus difficile l'application manuelle des mises à jour. Il est plutôt recommandé de remplacer la variable SERVER dans la configuration de mise à jour par une valeur invalide. On peut le configurer avec une configuration Container Linux (doit être transpilé vers Ignition JSON) :

```
storage:
  files:
    - path: /etc/flatcar/update.conf
      mode: 0644
      contents:
        inline: |
                    SERVER=disabled
```

Pour exécuter manuellement les mises à jour, supprimer le fichier et exécuter `update_engine_client -update` ou attendre que la mise à jour se produise. Une fois que le moteur de mise à jour a appliqué la mise à jour à la partition passive, on peut recréer le fichier pour désactiver les mises à jour automatiques.

L'outil **flatcar-update** supprime automatiquement la ligne **SERVER=disabled** pour appliquer une mise à jour manuelle et la restaure après l'application de la mise à jour (il dispose également d'un commutateur explicite **–disable-afterwards** pour définir **SERVER=disabled**) :

```
$ # Par exemple, mettre à jour vers la dernière version stable :
$ VER=$(curl -fsSL
https://stable.release.flatcar-linux.net/amd64-usr/current/version.txt |
grep FLATCAR_VERSION= | cut -d = -f 2)
$ sudo flatcar-update --to-version $VER
```

Si on n'a pas défini **SERVER=disabled**, il faut utiliser le commutateur **–disable-afterwards** pour définir **SERVER=disabled** dans `/etc/flatcar/update.conf` pour désactiver les mises à jour. La désactivation des mises à jour garantit que l'on restera sur la version spécifiée.

Après avoir appliqué la mise à jour, attendre que le redémarrage se produise ou l'invoquer manuellement.

Comme alternative, on peut masquer les services **update-engine** et **locksmithd** comme suit :

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

Configurations Linux du conteneur

Par défaut, il n'y a pas de mot de passe ni aucun autre moyen de se connecter à un nouveau système **Flatcar Container Linux**. Le moyen le plus simple de configurer des comptes, d'ajouter des unités systemd, etc., consiste à utiliser Container Linux Configs.

Après avoir utilisé Container Linux Config Transpiler pour produire une configuration Ignition, le script d'installation traitera le fichier `ignition.json` spécifié avec l'indicateur **-i** et l'utilisera lors du démarrage de l'installation.

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

```
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**.

Pour démarrer le script d'installation avec une référence à notre configuration Ignition, exécuter :

```
flatcar-install -d /dev/sda -C stable -i ~/ignition.json
```

L'exemple suivant configurera les composants **Flatcar Container Linux** : **etcd** et **flannel**.
Remplacer **<PEER_ADDRESS>** par l'adresse IP ou DNS de l'hôte.

```
passwd:
  users:
    - name: core
      ssh_authorized_keys:
        - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGdByTgSVHq.......
etcd:
  # generate a new token for each unique cluster from
https://discovery.etcd.io/new?size=3
  # specify the initial size of your cluster with ?size=X
  discovery: https://discovery.etcd.io/<token>
  advertise_client_urls:
http://<PEER_ADDRESS>:2379,http://<PEER_ADDRESS>:4001
  initial_advertise_peer_urls: http://<PEER_ADDRESS>:2380
  # listen on both the official ports and the legacy ports
  # legacy ports can be omitted if your application doesn't depend on them
  listen_client_urls: http://0.0.0.0:2379,http://0.0.0.0:4001
  listen_peer_urls: http://<PEER_ADDRESS>:2380
systemd:
  units:
    - name: flanneld.service
      enable: true
      dropins:
      - name: 50-network-config.conf
        contents: |
          [Service]
          ExecStartPre=/usr/bin/etcdctl set /kinvolk.io/network/config
'{"Network":"10.1.0.0/16", "Backend": {"Type": "vxlan"}}'
```

[1)](#) [2)](#)
,
Butane, anciennement **F**edora **C**oreOS **C**onfig **T**ranspiler, **FCCT**, traduit les configurations butane lisibles par l'homme en configurations **Ignition** lisibles par machine
[3)](#)
Pour exécuter un conteneur simple avec **Podman**, utiliser la syntaxe suivante: `podman run -d -p 8080:80 docker.io/library/nginx`, dans cet exemple, le port de conteneur **80/tcp** est mappé sur le port hôte **8080/tcp** pour permettre à **nginx** de répondre aux requêtes externes. On peut vérifier l'état avec la commande: `podman ps*`
[4)](#)
Par défaut le démon **Docker** est désactivé