Coreos: Utilisation avancée

Table of Contents

Processus de démarrage de CoreOS Container Linux

Le processus de démarrage de **Container Linux** est basé sur le processus de démarrage standard de Linux et, comme ce processus est déjà bien documenté et généralement bien compris, le présent document se concentre sur les aspects propres au démarrage de **Container Linux**.

Chargeur de démarrage

GRUB est le premier programme exécuté au démarrage d'un système **Container Linux**. La configuration de conteneur **GRUB Container Linux** a plusieurs rôles.

- Tout d’abord, la configuration **GRUB** spécifie la partition `usr` à utiliser parmi les deux partitions `usr` utilisées par **Container Linux** pour fournir des mises à niveau et des restaurations

automiques.
- Deuxièmement, **GRUB** recherche un fichier appelé **coreos/first_boot** dans la partition système EFI pour déterminer s'il s'agit du premier démarrage d'une machine. Si ce fichier est trouvé, GRUB définit le paramètre de ligne de commande du noyau `Linux coreos.first_boot=`. Le paramètre est utilisé dans les étapes ultérieures du processus de démarrage.
- Enfin, **GRUB** recherche le **GUID** de disque initial (`00000000-0000-0000-0000-000000000001`) intégré dans les images `Container Linux`. Ce **GUID** est randomisé ultérieurement dans le processus d'amorçage de sorte que les disques individuels puissent être identifiés de manière unique. Définir un autre paramètre de ligne de commande du noyau Linux, `coreos.randomize_guid=00000000-0000-0000-0000-000000000001`.

Espace utilisateur avancé

Après **GRUB**, le processus de démarrage de **Container Linux** est transféré dans le système de fichiers RAM initial: **initramfs** monte le système de fichiers racine, randomise le **GUID** du disque et exécute `Ignition`.

Si le paramètre de noyau **coreos.randomize_guid** est fourni, le disque avec le **GUID** spécifié se voit attribuer un nouveau **GUID** aléatoire.

Si le paramètre de noyau **coreos.first_boot** est fourni et que la configuration est différente de zéro, **Ignition** et **networkd** sont démarrés. **Networkd** utilisera **DHCP** pour configurer des adresses IP temporaires et des itinéraires afin qu'`Ignition` puisse extraire sa configuration du réseau.

Démarrage

Lorsque **Ignition** s'exécute sur **Container Linux**, il lit la ligne de commande Linux à la recherche de `coreos.oem.id`. `Ignition` utilise cet identificateur pour déterminer où lire la configuration fournie par l'utilisateur et quelle configuration spécifique au fournisseur combiner ce dernier avec l'utilisateur. La configuration spécifique effectue la configuration de base de la machine et peut inclure l'activation de `coreos-metadata-sshkeys@.service`.

Après l'exécution réussie de `Ignition`, si **coreos.first_boot** était défini sur la valeur spéciale détectée, **Ignition** monte la partition système EFI et supprime le fichier **coreos/first_boot**.

Espace utilisateur

Une fois que toutes les tâches dans **initramfs** sont terminées, la machine pivote dans l'espace utilisateur.C'est à ce stade que **systemd** démarre les unités, y compris, le cas échéant, `coreos-metadata-sshkeys@core.service`.

Clés SSH

`Coreos-metadata-sshkeys@core.service` est chargé d'extraire les clés SSH de l'environnement de la machine, qui sont écrites dans `~core/.ssh/registered_keys.d/coreos-metadata` et `update-ssh-keys` est exécuté pour mettre à jour `~core/.ssh/authorized_keys`. Sur les plates-formes cloud, les clés sont lues à partir du service de métadonnées du fournisseur. Ce service n'est pas pris en charge sur toutes les plates-formes et est activé par `Ignition` uniquement sur celles qui sont prises en charge.

Cluster etcd

CoreOS est une puissante distribution Linux conçue pour simplifier la gestion de déploiements de grande envergure et évolutifs sur une infrastructure variée. CoreOS est conçu pour la sécurité, la cohérence et la fiabilité. Au lieu d'installer des packages via yum ou apt, CoreOS utilise des conteneurs Linux pour gérer less services à un niveau d'abstraction plus élevé. Le code d'un seul service et toutes les dépendances sont regroupés dans un conteneur pouvant être exécuté sur un ou plusieurs ordinateurs CoreOS.

Pour démarrer un cluster **etcd** , il faut que chaque membre se connaisse. Dans certains cas, il est possible qu'on ne puisse déterminer les adresses IP des membres d'un cluster à l'avance. Dans ce cas, on peut amorcer un cluster **etcd** à l'aide d'un Service de découverte.

Cette section couvrira les mécanismes suivants pour amorcer un cluster **etcd**:

- Statique
- Etcd Discovery

Cluster statique

Description du cluster

L'exemple décrit un cluster de trois machines, etc., avec les détails suivants:

Nom	Adresse	Nom d'hôte
Infra0	10.0.1.10	infra0.example.com
Infra1	10.0.1.11	infra1.example.com
Infra2	10.0.1.12	infra2.example.com

Comme on connaît les membres du cluster, leurs adresses et la taille du cluster avant de démarrer, on peut utiliser une configuration d'amorçage hors connexion en définissant l'indicateur de cluster initial. Chaque ordinateur obtiendra la ligne de commande ou les variables d'environnement suivantes:

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2
```

```
380,infra2=http://10.0.1.12:2380"
ETCD_INITIAL_CLUSTER_STATE=new
```

```
--initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0
.1.12:2380 \
--initial-cluster-state new
```

les URL spécifiées dans **initial-cluster** sont les URL homologues annoncées, c'est-à-dire qu'elles doivent correspondre à la valeur de **initial-advertise-peer-urls** sur les nœuds respectifs.

Création du cluster

Pour créer plusieurs groupes (ou créer et détruire un seul groupe) avec la même configuration à des fins de test, il est vivement recommandé de spécifier un jeton de cluster initial unique pour les différents clusters. Ainsi, **etcd** peut générer des identifiants de grappe et des identifiants de membre uniques pour les grappes, même s'ils ont exactement la même configuration, ce qui peut vous protéger de l'interaction entre grappes, qui pourrait corrompre vos grappes.

etcd écoute dans **listen-client-urls** pour accepter le trafic client. Le membre **etcd** annonce les URL spécifiées dans **advertise-client-urls** les autres membres, mandataires, clients. Il faut s'assurer que **advertise-client-urls** sont accessibles depuis les clients cibles. Il ne faut pas définir **advertise-client-urls** sur **localhost** ou le laisser par défaut lorsqu'on souhaite que les clients distants accèdent à **etcd**.

Sur chaque machine, lancer etcd avec ces drapeaux:

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0
.1.12:2380 \
  --initial-cluster-state new
```

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
  --listen-peer-urls http://10.0.1.11:2380 \
  --listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.11:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster
```

```
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0
.1.12:2380 \
  --initial-cluster-state new
```

```
$ etcd --name infra2 --initial-advertise-peer-urls http://10.0.1.12:2380 \
  --listen-peer-urls http://10.0.1.12:2380 \
  --listen-client-urls http://10.0.1.12:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.12:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0
.1.12:2380 \
  --initial-cluster-state new
```

Les paramètres de ligne de commande commençant par **–initial-cluster** seront ignorés lors des exécutions suivantes d'**etcd**. On peut supprimer les variables d'environnement ou les indicateurs de ligne de commande après le processus d'amorçage initial.

Cas d'erreur

Dans l'exemple suivant, on n'a pas inclus un nouvel hôte dans la liste des nœuds énumérés: s'il s'agit d'un nouveau cluster, le nœud doit être ajouté à la liste des membres du cluster initial.

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
  --listen-peer-urls https://10.0.1.11:2380 \
  --listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.11:2379 \
  --initial-cluster infra0=http://10.0.1.10:2380 \
  --initial-cluster-state new
etcd: infra1 not listed in the initial cluster config
exit 1
```

Si on configure un homologue avec un ensemble de configuration différent et qu'on tente de rejoindre ce cluster, on obtiendra une incompatibilité ID de cluster et **etcd** se fermera.

```
$ etcd --name infra3 --initial-advertise-peer-urls http://10.0.1.13:2380 \
  --listen-peer-urls http://10.0.1.13:2380 \
  --listen-client-urls http://10.0.1.13:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.13:2379 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra3=http://10.0
.1.13:2380 \
  --initial-cluster-state=new
etcd: conflicting cluster ID to the target cluster (c6ab534d07e8fcc4 !=
bc25ea2a74fb18b0). Exiting.
exit 1
```

Dans cet exemple, on essaye de mapper un nœud (infra0) sur une adresse différente (127.0.0.1:2380) de son adresse énumérée dans la liste des grappes (10.0.1.10:2380). Si ce nœud doit écouter plusieurs adresses , toutes les adresses doivent être reflétées dans la configuration “grappe initiale”

```
$ etcd --name infra0 --initial-advertise-peer-urls http://127.0.0.1:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --initial-cluster
infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0
.1.12:2380 \
  --initial-cluster-state=new
etcd: error setting up initial cluster: infra0 has different advertised URLs
in the cluster and advertised peer URLs list
exit 1
```

Service etcd discovery personnalisé

Dans un certain nombre de cas, il est possible qu'on ne connaisse pas à l’avance les adresses IP des homologues du cluster, ce qui est fréquent lorsqu'on utilise des fournisseurs de cloud ou lorsque le réseau utilise DHCP. Dans ces cas, au lieu de spécifier une configuration statique, on peut utiliser un serveur existant. `Etc cluster` pour amorcer un nouveau processus, qu'on appelle **discovery**.

Deux méthodes peuvent être utilisées pour la découverte:

- Service de découverte Etcd
- Enregistrements DNS SRV

Un service de découverte, `https://discovery.etcd.io`, est fourni gratuitement pour aider à connecter des instances **etcd**. On peut les générer très facilement:

```
$ curl -w "\n" 'https://discovery.etcd.io/new?size=3'
https://discovery.etcd.io/6a28e078895c5ec737174db2419bb2f3
```

L'URL de découverte peut être fournie à chaque machine Linux conteneur via les configurations de conteneur Linux.

Mais ce service stocke un tas d'information: liste d'adresses homologues, métadonnées et taille initiale du cluster sous une adresse unique, appelée URL de découverte.

Cette section décrit la création d'un service `etcd discovery` auto-hébergé.

Création du service

Activer du service

```
$ etcd -name discovery -addr 127.0.0.1:4000 -peer-addr 127.0.0.1:7000
```

Générer un token avec uuidgen

```
$ export UUID=$(uuidgen)
$ echo $UUID
40134540-b53c-46b3-b34f-33b4f0ae3a9c
```

Test du service

Démarrer les nodes

```
$ etcd -name node1 --addr 127.0.0.1:4001 --peer-addr 127.0.0.1:7001 \
-discovery http://127.0.0.1:4000/v2/keys/$UUID
```

```
$ etcd -name node2 --addr 127.0.0.1:4002 --peer-addr 127.0.0.1:7002 \
-discovery http://127.0.0.1:4000/v2/keys/$UUID
```

```
$ etcd -name node3 --addr 127.0.0.1:4003 --peer-addr 127.0.0.1:7003 \
-discovery http://127.0.0.1:4000/v2/keys/$UUID
```

Cela obligera chaque membre à s'inscrire lui-même auprès du service de découverte personnalisé **etcd** et à démarrer le cluster une fois que toutes les machines auront été enregistrées.

Vérifier l'état du cluster

```
$ curl localhost:7001/v2/admin/machines
[{"name":"node1","state":"leader","clientURL":"http://127.0.0.1:4001","peerU
RL":"http://127.0.0.1:7001"},{"name":"node2","state":"follower","clientURL":
"http://127.0.0.1:4002","peerURL":"http://127.0.0.1:7002"},{"name":"node3","
state":"follower","clientURL":"http://127.0.0.1:4003","peerURL":"http://127.
0.0.1:7003"}]
```

Vérifier l'état du service

```
$ curl localhost:4000/v2/keys/40134540-b53c-46b3-
b34f-33b4f0ae3a9c?recursive=true
{"action":"get","node":{"key":"/40134540-b53c-46b3-
b34f-33b4f0ae3a9c","dir":true,"nodes":[{"key":"/40134540-b53c-46b3-
```

```
b34f-33b4f0ae3a9c/node1","value":"http://127.0.0.1:7001","expiration":"2014-
10-31T23:05:50.268605238Z","ttl":604595,"modifiedIndex":19,"createdIndex":19
},{"key":"/40134540-b53c-46b3-
b34f-33b4f0ae3a9c/node2","value":"http://127.0.0.1:7002","expiration":"2014-
10-31T23:05:55.335074504Z","ttl":604600,"modifiedIndex":21,"createdIndex":21
},{"key":"/40134540-b53c-46b3-
b34f-33b4f0ae3a9c/node3","value":"http://127.0.0.1:7003","expiration":"2014-
10-31T23:06:11.718215655Z","ttl":604616,"modifiedIndex":22,"createdIndex":22
}],"modifiedIndex":19,"createdIndex":19}}
```


Utilisation du service

Discovery utilise un cluster existant pour s’amorcer lui-même. Lorsqu'on utilise un service **discovery** auto hébergé, on peut créer une URL de la manière suivante:

```
$ curl -X PUT http://127.0.0.1:4000/v2/keys/{UUID}/_config/size -d value=3
```

En définissant la clé de taille sur l'URL, on créer une URL de découverte avec une taille de cluster attendue de 3.

Lorsqu'on ammorce un cluster **etcd** à l'aide du service de découverte avec un nombre de membres **etcd** supérieur au nombre attendu, les processus supplémentaires **etcd** redeviendront des mandataires par défaut.

Chaque membre doit avoir un indicateur de nom différent spécifié. Un nom d'hôte ou un identifiant d'ordinateur peut être un bon choix. Ou la découverte échouera à cause d'un nom dupliqué.

Maintenant, on peut démarrer **etcd** avec ces drapeaux pertinents pour chaque membre:

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0f
b83
```

```
$ etcd --name infra1 --initial-advertise-peer-urls http://10.0.1.11:2380 \
  --listen-peer-urls http://10.0.1.11:2380 \
  --listen-client-urls http://10.0.1.11:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.11:2379 \
  --discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0f
b83
```

```
$ etcd --name infra2 --initial-advertise-peer-urls http://10.0.1.12:2380 \
  --listen-peer-urls http://10.0.1.12:2380 \
  --listen-client-urls http://10.0.1.12:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.12:2379 \
  --discovery
https://myetcd.local/v2/keys/discovery/6c007a14875d53d9bf0ef5a6fc0257c817f0fb83
```