CoreOS: Lab 1 Configuration d'un cluster

Objet	Création d'un cluster CoreOS
Niveau requis	débutant, avisé
Débutant, à savoir	
Suivi	:DRAFT:

Ce lab explique comment amorcer un nouveau cluster de système d'exploitation Production Core en tant que service haute disponibilité avec etcd2, Fleet, Flannel, Confd, Nginx Balancer et Docker.

Les principaux composants de CoreOS - etc, Docker et systemd.

- **etcd**: magasin de valeurs-clés pour l'enregistrement et la découverte de services
- **fleet**: planification et basculement des conteneurs Docker sur un cluster CoreOS
- **flannel**: attribue à chaque conteneur docker une adresse IP unique vous permettant d'accéder au port interne (c'est-à-dire le port 80 et non le 32679)
- **confd**: surveille etcd pour les noeuds arrivant / partant et met à jour (avec rechargement) la configuration de nginx en utilisant le modèle spécifié

Configuration de base

Générer un nouveau jeton pour le cluster: `https://discovery.etcd.io/new?size=X`, où X correspond au nombre de serveurs

Etape1 : Editer le fichier Cloud Config

```
sudo vi /var/lib/coreos-install/user_data

coreos:
  etcd2:
# Génére un nouveau jeton pour chaque cluster unique à partir de
https://discovery.etcd.io/new
# discovery: https://discovery.etcd.io/<jeton>
discovery: https://discovery.etcd.io/9c19239271bcd6be78d4e8acfb393551
# Les déploiements multi-régions et multi-cloud doivent utiliser $
public_ipv4
advertise-client-urls: http://$private_ipv4:2379,http://$private_ipv4:4001
initial-advertise-peer-urls: http://$private_ipv4:2380
# Ecoute à la fois sur les ports officiels et les ports existants
# Les ports hérités peuvent être omis si votre application n'en dépend pas
listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0::4001
listen-peer-urls: http: // $ private_ipv4: 2380

  fleet:
public-ip: $private_ipv4
```

```
metadata: region=europe,public_ip=$public_ipv4

  flannel:
interface: $private_ipv4

  units:
- name: etcd2.service
  command: start
  # See issue:
https://github.com/coreos/etcd/issues/3600#issuecomment-165266437
  drop-ins:
    - name: "timeout.conf"
      content: |
        [Service]
        TimeoutStartSec=0
- name: fleet.service
  command: start

# Network configuration should be here, e.g:
# - name: 00-eno1.network
#   content:
"[Match]\nName=eno1\n\n[Network]\nDHCP=yes\n\n[DHCP]\nUseMTU=9000\n"
# - name: 00-eno2.network
#   runtime: true
#   content:
"[Match]\nName=eno2\n\n[Network]\nDHCP=yes\n\n[DHCP]\nUseMTU=9000\n"
- name: flanneld.service
  command: start
  drop-ins:
  - name: 50-network-config.conf
    content: |
      [Service]
      ExecStartPre=/usr/bin/etcdctl set /coreos.com/network/config '{
"Network": "10.1.0.0/16" }'
- name: docker.service
  command: start
  drop-ins:
  - name: 60-docker-wait-for-flannel-config.conf
    content: |
      [Unit]
      After=flanneld.service
      Requires=flanneld.service

      [Service]
      Restart=always
- name: docker-tcp.socket
  command: start
  enable: true
  content: |
    [Unit]
    Description=Docker Socket for the API
```

```
    [Socket]
    ListenStream=2375
    Service=docker.service
    BindIPv6Only=both

    [Install]
    WantedBy=sockets.target
```

Ajouter ces lignes à Cloud Config pour que le réseau privé fonctionne:

```
units:
  # ...
  - name: 00-eno2.network
runtime: true
content:
"[Match]\nName=eno2\n\n[Network]\nDHCP=yes\n\n[DHCP]\nUseMTU=9000\n"
```

Etape 2: Valider les modifications

```
sudo coreos-cloudinit -validate --from-file /var/lib/coreos-
install/user_data
```

Etape 3: Redémarrer le système

```
sudo reboot
```

Etape 4: Vérifier l'état de etcd2

```
sudo systemctl status -r etcd2
```

La sortie doit contenir une ligne suivante:

```
 Active: active (running)
```

Parfois cela prend du temps, Attendre quelques minutes.

Répéter les étapes 1 à 4 pour chaque serveur du cluster.

Etape 5: Vérifier la santé du cluster et statut de fleet

```
# devrait être en bonne santé
sudo etcdctl cluster-health
# devrait afficher tous les serveurs
sudo fleetctl list-machines
```

Tester le cluster

Création d'unités dans fleet

Dans le répertoire personnel:

```
cd ~
```

Créer une nouvelle unité de modèle d'application. Par exemple, `vi test-app@.service` et ajouter les lignes suivantes:

```
[Unit]
Description=test-app%i
After=docker.service

[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill test-app%i
ExecStartPre=-/usr/bin/docker rm test-app%i
ExecStartPre=/usr/bin/docker pull willrstern/node-sample
ExecStart=/usr/bin/docker run -e APPNAME=test-app%i --name test-app%i -P
willrstern/node-sample
ExecStop=/usr/bin/docker stop test-app%i
```

Soumettre une unité de modèle de demande à fleet

```
fleetctl submit test-app@.service
```

Démarrer de nouvelles instances

A partir de l'unité de modèle d'application:

```
fleetctl start test-app@1
```

```
fleetctl start test-app@2
fleetctl start test-app@3
```

Vérifier que tout fonctionne bien

Vérifier que toutes les instances ont été démarrées et actives. Cela pourrait prendre quelques minutes.

```
$ fleetctl list-units
UNIT            MACHINE                 ACTIVE    SUB
test-app@1.service    e1512f34.../10.1.9.17    active    running
test-app@2.service    a78a3229.../10.1.9.18    active    running
test-app@3.service    081c8a1e.../10.1.9.19    active    running
Configurer les règles du pare-feu
```

Equilibreurs de charge et découverte de service

Créer les fichiers someapp

Créer les fichiers `someapp@.service`, `someapp-discovery@.service` et `someapp-lb@.service` avec les contenus suivants.

someapp@.service

```
[Unit]
Description=someapp%i
Requires=docker.service
After=docker.service

[Service]
# Let the process take awhile to start up (for first run Docker containers)
TimeoutStartSec=0

# Directives with "=-" are allowed to fail without consequence
ExecStartPre=-/usr/bin/docker kill someapp%i
ExecStartPre=-/usr/bin/docker rm someapp%i
ExecStartPre=/usr/bin/docker pull denisizmaylov/node-sample
ExecStart=/usr/bin/docker run -e APPNAME=someapp%i --name someapp%i -P
denisizmaylov/node-sample
ExecStop=/usr/bin/docker stop someapp%i
```

someapp-discovery@.service

```
[Unit]
Description=Announce Someapp%i

# Requirements
BindsTo=someapp@%i.service
Requires=etcd2.service
Requires=docker.service

# Dependency ordering
After=someapp@%i.service
After=etcd2.service
After=docker.service

[Service]
ExecStart=/bin/sh -c "while true; do etcdctl set
/services/someapp/upstream/someapp%i \"$(sleep 5 && docker inspect -f
'{{.NetworkSettings.IPAddress}}' someapp%i):3000\" --ttl 60;sleep 45;done"
ExecStop=/usr/bin/etcdctl rm /services/someapp/upstream/someapp%i

[X-Fleet]
MachineOf=someapp@%i.service
```

someapp-lb@.service

```
[Unit]
Description=someapp-lb%i

# Requirements
Requires=docker.service
Requires=etcd2.service

# Dependency ordering
After=docker.service
After=etcd2.service

[Service]
# Let the process take awhile to start up (for first run Docker containers)
TimeoutStartSec=0

# Change killmode from "control-group" to "none" to let Docker remove
# work correctly.
KillMode=none

# Get CoreOS environmental variables
EnvironmentFile=/etc/environment

# Directives with "=-" are allowed to fail without consequence
```

```
ExecStartPre=-/usr/bin/docker kill someapp-lb%i
ExecStartPre=-/usr/bin/docker rm someapp-lb%i
ExecStartPre=/usr/bin/docker pull denisizmaylov/nginx-lb
ExecStart=/usr/bin/sh -c "/usr/bin/docker run --name someapp-lb%i --rm -p
80:80 -e SERVICE_NAME=someapp -e ETCD=\"$(ifconfig docker0 | awk
'/\\<inet\\>/ { print $2 }'):2379\" denisizmaylov/nginx-lb"
ExecStop=/usr/bin/docker stop someapp-lb%i

[X-Fleet]
Conflicts=someapp-lb@*
MachineMetadata=loadbalancer=true
```

Modifier ces modèles d'unités en fonction de la configuration.

Soumettre ces fichiers à fleet

```
fleetctl submit someapp@.service
fleetctl submit someapp-discovery@.service
fleetctl submit someapp-lb@.service
```

Démarrer des instances d'unités

```
fleetctl start someapp@{1..6}
fleetctl start someapp-discovery@{1..6}
fleetctl start someapp-lb@{1..2}
```

Vérifier que tout fonctionne bien

```
fleetctl list-units
```

Dépannage

Un service ne fonctionne pas

Utiliser ces commandes pour déboguer:

```
# aussi pour fleet, etcd, flanneld
sudo systemctl start etcd2
sudo systemctl status etcd2
sudo journalctl -xe
sudo journalctl -xep3
```

```
sudo journalctl -ru etcd2
```

Afficher l'état des unités

`fleet list-unit` affiche l'état d'échec pour toutes les unités

- Pour les unités locales:

```
sudo fleetctl journal someapp@1
```

- Pour les unités distantes:

```
fleetctl journal someapp@1
```

SSH_AUTH_SOCK n'est pas définie

Lorsque `fleetctl` reponds avec:

```
Erreur lors de l'exécution de la commande à distance: la variable
d'environnement SSH_AUTH_SOCK n'est pas définie. Vérifiez que ssh-agent est
en cours d'exécution.
```

- Vérifier la connection avec `ssh -A`.
- n'utiliser pas sudo pour des machines distantes. Car un processus sous sudo ne peut pas accéder à SSH_AUTH_SOCK.

Nom déjà utilisé

Lorsque le démon réponsd:

```
Conflict. Le nom "someapp1" est déjà utilisé par le conteneur c4acbb70c654.
```

il faut supprimer (ou renommer) ce conteneur pour pouvoir réutiliser ce nom.

```
fleetctl stop someapp@1
docker rm someapp1
fleetctl start someapp@1
```

La commande ssh de fleet ne fonctionne pas

Il faut:

- s'assurer que la clé publique a été ajoutée partout dans `user_data`. Sur chaque serveur.

- se connecter au serveur avec l'agent SSH:

```
eval `ssh-agent -s`
ssh-add ~/.ssh/id_rsa
ssh -A <your-host>
```