Mise en situation professionnelle	**EasyLinux**
	Cours : Kubernetes
Sujet : Sécurisation du cluster	Numéro : 20 à 22 Version : 1.0

Objectifs :

Sécuriser le cluster

Prérequis :

aucun

Principales tâches à réaliser :

Sécurisation du cluster

20 Création d’un namespace

Créer un namespace Dev

```
apiVersion: v1
kind: Namespace
metadata:
   name: dev
```

21 Ajout utilisateur

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: developpeur
  namespace: dev
```

22 Droits d’accès application

Créer un rôle qui donne tous les droits sur le namespace

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: dev-user-full-access
  namespace: dev
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["*"]
- apiGroups: ["batch"]
  resources:
  - jobs
  - cronjobs
  verbs: ["*"]
```

Assigner ce rôle à l'utilisateur

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: mynamespace-user-view
  namespace: dev
subjects:
- kind: ServiceAccount
  name: developpeur
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: dev-user-full-access
```

Créer l'utilisateur, le rôle et le rôleBinding

```
kubectl create -f
```

22.1 Utilisation

Rechercher le nom du secret auto-généré par Kubernetes (de la formet developpeur-token-xxx)

```
kubectl describe sa developpeur -n dev
```

Récupérer maintenant le token associé ainsi que le certificat

```
kubectl get secret developpeur-token-xxxxx -n dev -o "jsonpath={.data.token}" | base64 -
D
```

Pour le certificat

```
kubectl get secret developpeur-token-xxxxx -n dev -o "jsonpath={.data['ca\.crt']}"
```

Pour utiliser le cluster, il faut configurer kubectl, il faut créer un répertoire .kube et générer un fichier config avec ce contenu :

```
apiVersion: v1
kind: Config
preferences: {}

# Define the cluster
clusters:
- cluster:
    certificate-authority-data: PLACE CERTIFICATE HERE
    # You'll need the API endpoint of your Cluster here:
    server: https://YOUR_KUBERNETES_API_ENDPOINT
  name: my-cluster

# Define the user
users:
- name: developpeur
  user:
    as-user-extra: {}
    client-key-data: PLACE CERTIFICATE HERE
    token: PLACE USER TOKEN HERE

# Define the context: linking a user to a cluster
contexts:
- context:
    cluster: my-cluster
    namespace: dev
    user: developpeur
  name: dev

# Define current context
current-context: dev
```

23 Remplacer un noeud

24 Ajouter un master

Copier le certificat sur le futur master

```
scp root@<kubemaster01-ip-address>:/etc/kubernetes/pki/* /etc/kubernetes/pki
```

Créer un loadbalancer devant vos nœuds master. La façon de procéder dépend de l'environnement : un fournisseur de cloud computing Load Balancer, utiliser NGINX, keepalived ou HAproxy.

Pour démarrer, utilisez le fichier config.yaml :

```
apiVersion: kubeadm.k8s.io/v1alpha1
kind: MasterConfiguration
api:
  advertiseAddress: <private-ip>
etcd:
  endpoints:
 - https://<your-ectd-ip>:2379
  caFile: /etc/kubernetes/pki/etcd/ca.pem
  certFile: /etc/kubernetes/pki/etcd/client.pem
  keyFile: /etc/kubernetes/pki/etcd/client-key.pem
networking:
  podSubnet: <podCIDR>
apiServerCertSANs:
- <load-balancer-ip>
apiServerExtraArgs:
  apiserver-count: "2"
```

Avec :

- `your-ectd-ip` l'adresse du ETCD
- `private-ip` l'adresse IP du serveur Master.
- `<podCIDR>` le CIDR utilisé lors du kubeadm init
- `load-balancer-ip` l'adresse du loadbalancer

Lancer ensuite :

```
kubeadm init --config=config.yaml
```

25 Sauvegarder un master

Kubeadm est une boîte à outils de base qui vous aide à démarrer un cluster Kubernetes simple. Il est destiné à servir de base à des outils de déploiement de plus haut niveau, comme les playbooks Ansible. Un cluster Kubernetes typique configuré avec kubeadm se compose d'un seul maître Kubernetes, qui est la machine coordonnant le cluster, et de plusieurs nœuds Kubernetes, qui sont les machines exécutant la charge de travail réelle.

La gestion des défaillances de nœuds est simple : Lorsqu'un nœud tombe en panne, le maître détecte la panne et reprogramme la charge de travail aux autres nœuds. Pour revenir au nombre de nœuds souhaité, vous pouvez simplement créer un nouveau nœud et l'ajouter au cluster. Pour ajouter un nouveau nœud à un cluster existant, vous créez d'abord un jeton sur le maître avec kubeadm token create, puis vous utilisez ce jeton sur le nouveau nœud pour rejoindre le cluster avec kubeadm join.

Faire face à l'échec du master est plus compliqué. La bonne nouvelle, c'est que L'échec du maître n'est pas aussi grave qu'il n'y paraît. Le cluster et toutes les charges de travail continueront à fonctionner avec exactement la même configuration qu'avant l'échec. Les applications s'exécutant dans le cluster Kubernetes seront toujours utilisables. Cependant, il ne sera pas possible de créer de nouveaux déploiements ou de récupérer des pannes de nœuds sans le master.

26 Cluster HA vs Sauvegarde/Restauration

Une façon de faire face aux échecs du master est de mettre en place un cluster à haute disponibilité. L'idée est de mettre en place un cluster etcd répliqué, et d'exécuter une instance etcd avec chaque instance master. De cette façon, aucune donnée ne sera perdue si une seule instance master échoue.

L'approche HA n'est pas toujours meilleure qu'une configuration à master unique :

- La surveillance d'un seul master est beaucoup plus simple que la surveillance d'un cluster etcd répliqué. L'exécution d'un cluster etcd répliqué ne vous dispense pas de la nécessité de mettre en place une surveillance.
- Le temps de récupération n'est pas nécessairement plus rapide lorsqu'on compare une configuration HA avec une configuration mono maître. Dans une configuration à master unique, l'outil de surveillance détectera les défaillances du master et déclenchera un script de restauration automatique. Ce n'est pas nécessairement plus lent que le mécanisme de basculement de etcd.
- La configuration HA ne vous empêche pas d'implémenter la sauvegarde/restauration pour le master, car il est toujours possible que vous détruisiez accidentellement les données dans un cluster etcd répliqué.

Le principal inconvénient de l'approche de sauvegarde/restauration est qu'il n'y a pas de sauvegarde en temps réel de l'état du cluster. Comme indiqué ci-dessous, nous utiliserons un CronJob

Kubernetes pour créer des sauvegardes du cluster etcd. Lorsque le master échoue, tous les changements dans la configuration du cluster après la dernière exécution de CronJob sont perdus. Le cluster sera restauré exactement dans le même état (déploiements, services, etc.) que lors de l'exécution du dernier CronJob. Bien que vous puissiez exécuter le CronJob toutes les minutes, vous n'obtiendrez pas la sauvegarde en temps réel qu'un cluster etcd répliqué fournit.

La pertinence d'un déploiement d'HA dépend de la façon dont vous utilisez le cluster. Si vous changez la configuration du cluster très fréquemment (déployez des applications si souvent que vous avez besoin d'une sauvegarde en temps réel de l'état du cluster), vous pourriez bénéficier de la duplication des données etcd dans une configuration HA. Si votre cluster change à un rythme plus lent, l'approche de sauvegarde/restauration pourrait être la meilleure option car elle simplifie les opérations.

27 Données à sauvegarder

Deux éléments de données à sauvegarder :

- Les fichiers de certificat racine /etc/kubernetes/pki/ca.crt et /etc/kubernetes/pki/ca.key.
- Les données etcd.

La sauvegarde du certificat racine est une opération unique que vous effectuerez manuellement après avoir créé le master avec kubeadm init. Le reste traite de la façon de sauvegarder les données etcd.

27.1 Sauvegarder Etcd

Un CronJob Kubernetes pour sauvegarder les données etcd

Comme indiqué dans la documentation etcd, vous créez une sauvegarde des données etcd avec

```
ETCDCTL_API=3
etcdctl --endpoints $ENDPOINT snapshot save snapshot.db
```

Nous allons créer un CronJob Kubernetes pour exécuter cette commande périodiquement. Il n'est pas nécessaire d'installer etcdctl sur le système hôte ou de configurer un travail cron sur le système hôte.

L'en-tête, nous voulons exécuter le CronJob dans l'espace de nommage kube-system :

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: backup
  namespace: kube-system
```

Exemple un lancement toutes les trois minutes :

```
spec:
  schedule: "*/3 * * * *"
  jobTemplate:
    spec:
```

Un master Kubernetes utilise un module statique pour exécuter etcd. Nous réutilisons l'image du docker etcd de ce pod pour le CronJob. La définition du pod se trouve dans /etc/kubernetes/manifests/etcd.yaml.

```
      template:
        spec:
          containers:
          - name: backup
            # Same image as in /etc/kubernetes/manifests/etcd.yaml
            image: k8s.gcr.io/etcd-amd64:3.1.12
```

La commande ci-dessous créera des fichiers de sauvegarde du type :

/backup/etcd-snapshot-2018-05-24_21:54:03_UTC.db

Les paramètres additionnels à etcdctl spécifient les certificats et l'URL auxquels accéder etcd.

```
            env:
            - name: ETCDCTL_API
              value: "3"
            command: ["/bin/sh"]
            args: ["-c", "etcdctl --endpoints=https://127.0.0.1:2379
--cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/healthcheck-
client.crt --key=/etc/kubernetes/pki/etcd/healthcheck-client.key snapshot save
/backup/etcd-snapshot-$(date +%Y-%m-%d_%H:%M:%S_%Z).db"]
```

La commande etctctl ci-dessus suppose que les volumes suivants ont été mappés dans le conteneur Docker :

- /etc/kubernetes/pki/etcd : Le chemin sur le système hôte où les informations d'identification etcd sont stockées.
- /Backup : Le volume persistant où les sauvegardes sont créées.

Il faudra monter les volumes :

```
            volumeMounts:
            - mountPath: /etc/kubernetes/pki/etcd
              name: etcd-certs
              readOnly: true
            - mountPath: /backup
              name: backup
```

L'API etcd est disponible sur le port 2379 du système hôte. Nous devrons exécuter le CronJob dans l'espace de noms du réseau hôte pour qu'il puisse accéder à etcd en utilisant l'adresse IP loopback 127.0.0.1.

```
          hostNetwork: true
```

Normalement, Kubernetes empêche la planification des pod sur le master. Toutes les charges de travail sont exécutées sur les nœuds. Pour forcer le CronJob à s'exécuter sur le maître, nous spécifions un nodeSelector (voir affinité des nœuds), et nous spécifions que le CronJob doit "tolérer" l'effet NoSchedule qui empêche l'exécution des charges sur le maaster.

```
          nodeSelector:
            kubernetes.io/hostname: kube-master
          tolerations:
          - effect: NoSchedule
            operator: Exists
          restartPolicy: OnFailure
```

Enfin, nous devons définir les volumes utilisés dans le volumeMount. Le premier volume est le chemin sur le système hôte où les informations d'identification etcd sont stockées. Ces informations d'identification sont créées lorsque le cluster est configuré avec kubeadm init.

```
          volumes:
          - name: etcd-certs
            hostPath:
              path: /etc/kubernetes/pki/etcd
              type: DirectoryOrCreate
```

Le second volume est un volume persistant où la sauvegarde sera stockée. Ici, on va monter un partage CIFS donc j'utilise le plugin fstab/cifs. Cependant, il est possible d’utiliser un autre type de volume persistant, selon l'endroit où vous souhaitez stocker la sauvegarde.

```
          - name: backup
            flexVolume:
              driver: "fstab/cifs"
              fsType: "cifs"
              secretRef:
                name: "backup-volume-credentials"
              options:
                networkPath: "//my-server.com/backup"
                mountOptions: "dir_mode=0755,file_mode=0644,noperm"
```

Pour créer le CronJob, créer un fichier backup-cron-job.yml et exécuter

```
$ kubectl apply -f backup-cron-job.yml
```

Le plugin fstab/cifs nécessite un secret, c'est-à-dire le nom d'utilisateur et le mot de passe pour monter le volume CIFS. Vous n'en avez pas besoin si vous utilisez un autre type de volume persistant. Le secret est défini comme suit (nom d'utilisateur et mot de passe sont encodés en base64) :

```
apiVersion: v1
kind: Secret
metadata:
  name: backup-volume-credentials
  namespace: kube-system
type: fstab/cifs
data:
  username: 'ZXhhbXBsZQ=='
  password: 'bXktc2VjcmV0LXBhc3N3b3Jk'
```

Le fichier backup-volume-secret.yml est proposé ci-dessus, exécuter

```
$ kubectl apply -f backup-volume-secret.yml
```

NB : Le secret est spécifique au plugin volume fstab/cifs utilisé ici.

28 Restaurer un master

En cas de panne, créer un nouveau maître et l'initialiser avec les données de la sauvegarde. Avant d'exécuter kubeadm init sur le nouveau maître, il faut restaurer les données de la sauvegarde.

Commencer à restaurer les fichiers de certificat racine /etc/kubernetes/pki/ca.crt et /etc/kubernetes/pki/ca.key. Les permissions à donner sont 0644 pour ca.crt et 0600 pour ca.key.

Ensuite exécuter etcdctl pour restaurer la sauvegarde etcd. Il n'est pas indispensable d'installer etcdctl sur le système hôte, car on peut utiliser l'image Docker.

Supposons que la dernière sauvegarde soit stockée dans /mnt/etcd-snapshot-2018-05-24_21:54:54:03_UTC.db, Lancer :

```
mkdir -p /var/lib/etcd
docker run --rm \
    -v '/mnt:/backup' \
    -v '/var/lib/etcd:/var/lib/etcd' \
    --env ETCDCTL_API=3 \
    'k8s.gcr.io/etcd-amd64:3.1.12' \
    /bin/sh -c "etcdctl snapshot restore '/backup/etcd-snapshot-2018-05-
24_21:54:03_UTC.db' ; mv /default.etcd/member/ /var/lib/etcd/"
```

La commande ci-dessus crée un répertoire /var/lib/etcd/member/ avec les permissions 0700.

Enfin, lancer kubeadm init pour créer le nouveau master. Cependant, nous avons besoin d'un paramètre supplémentaire pour qu'il accepte les données etcd existantes :

```
kubeadm init –ignore-preflight-errors=DirAvailable--var-lib-etcd
```

En supposant que le nouveau master est accessible sous la même adresse IP ou le même nom d'hôte que l'ancien master, les nœuds se reconnecteront et le cluster sera à nouveau opérationnel.

29 Haute disponibilité

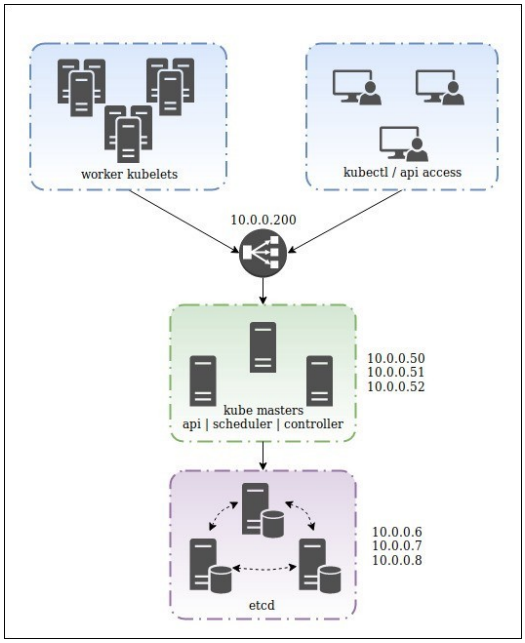
Quelques serveurs, un répartiteur de charge et kubeadm sont les seuls outils nécessaires à la configuration. Il est configuré à l'aide du loadbalancer pour la communication entre noeud (kubelet), et kubectl. Vous pouvez réaliser cette configuration chez un hébergeur, ou en local avec HAproxy. Il est nécessaire de mettre en œuvre un loadbalancer entre les nœuds master. Le port :6443 est utilisé en tant que backend pour le loadbalancer.

```
etcd   1 address      = 10.0.0.6
etcd   2 address      = 10.0.0.7
etcd   3 address      = 10.0.0.8

master 1 address      = 10.0.0.50
master 2 address      = 10.0.0.51
master 3 address      = 10.0.0.52

load balancer address = 10.0.0.200
```

Soit le schéma :



29.1 Etape 1 - Déployez les serveurs

Avant de commencer, déployer les serveurs ont été déployés avec les prérequis nécessaires (2Go, 2Cpu, pas de swap, connectivité réseau).

29.2 Étape 2 - Configuration etcd

Il y a plusieurs façons différentes de déployer etcd.

a Modifier le lancement de Kubelet

/etc/systemd/system/kubelet.service.d/20-etcd-service-manager.conf

```
[Service]
ExecStart=
ExecStart=/usr/bin/kubelet --address=127.0.0.1
--pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true
Restart=always
```

Puis lancer ce service

```
systemctl daemon-reload
systemctl restart kubelet
```

b Générer la configuration pour kubeadm

```
# Update HOST0, HOST1, and HOST2 with the IPs or resolvable names of your hosts
export HOST0=10.0.0.6
export HOST1=10.0.0.7
export HOST2=10.0.0.8

# Create temp directories to store files that will end up on other hosts.
mkdir -p /tmp/${HOST0}/ /tmp/${HOST1}/ /tmp/${HOST2}/

ETCDHOSTS=(${HOST0} ${HOST1} ${HOST2})
NAMES=("infra0" "infra1" "infra2")

for i in "${!ETCDHOSTS[@]}"; do
HOST=${ETCDHOSTS[$i]}
NAME=${NAMES[$i]}
cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml
apiVersion: "kubeadm.k8s.io/v1beta1"
kind: ClusterConfiguration
etcd:
    local:
        serverCertSANs:
        - "${HOST}"
        peerCertSANs:
        - "${HOST}"
        extraArgs:
            initial-cluster: ${NAMES[0]}=https://${ETCDHOSTS[0]}:2380,$
{NAMES[1]}=https://${ETCDHOSTS[1]}:2380,${NAMES[2]}=https://${ETCDHOSTS[2]}:2380
            initial-cluster-state: new
            name: ${NAME}
            listen-peer-urls: https://${HOST}:2380
            listen-client-urls: https://${HOST}:2379
            advertise-client-urls: https://${HOST}:2379
            initial-advertise-peer-urls: https://${HOST}:2380
EOF
done
```

c Créer une autorité de certification

S’il n’y a pas d’autorité de certification, la générer

```
kubeadm init phase certs etcd-ca
```

Cela crée 2 fichiers :

- /etc/kubernetes/pki/etcd/ca.crt
- /etc/kubernetes/pki/etcd/ca.key

d Créer un certificat pour chaque Noeud

```
kubeadm init phase certs etcd-server --config=/tmp/${HOST2}/kubeadmcfg.yaml
kubeadm init phase certs etcd-peer --config=/tmp/${HOST2}/kubeadmcfg.yaml
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST2}/kubeadmcfg.yaml
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST2}/kubeadmcfg.yaml
cp -R /etc/kubernetes/pki /tmp/${HOST2}/
# cleanup non-reusable certificates
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete

kubeadm init phase certs etcd-server --config=/tmp/${HOST1}/kubeadmcfg.yaml
kubeadm init phase certs etcd-peer --config=/tmp/${HOST1}/kubeadmcfg.yaml
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST1}/kubeadmcfg.yaml
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST1}/kubeadmcfg.yaml
cp -R /etc/kubernetes/pki /tmp/${HOST1}/
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete

kubeadm init phase certs etcd-server --config=/tmp/${HOST0}/kubeadmcfg.yaml
kubeadm init phase certs etcd-peer --config=/tmp/${HOST0}/kubeadmcfg.yaml
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST0}/kubeadmcfg.yaml
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST0}/kubeadmcfg.yaml
# No need to move the certs because they are for HOST0

# clean up certs that should not be copied off this host
find /tmp/${HOST2} -name ca.key -type f -delete
find /tmp/${HOST1} -name ca.key -type f -delete
```

e Copier le tout

```
USER=ubuntu
 HOST=${HOST1}
 scp -r /tmp/${HOST}/* ${USER}@${HOST}:
 ssh ${USER}@${HOST}
 USER@HOST $ sudo -Es
 root@HOST $ chown -R root:root pki
 root@HOST $ mv pki /etc/kubernetes/
```

f Valider

```
/tmp/${HOST0}
└── kubeadmcfg.yaml
---
/etc/kubernetes/pki
├── apiserver-etcd-client.crt
├── apiserver-etcd-client.key
└── etcd
    ├── ca.crt
    ├── ca.key
    ├── healthcheck-client.crt
    ├── healthcheck-client.key
    ├── peer.crt
    ├── peer.key
    ├── server.crt
    └── server.key
```

g Créer les manifests pour les pod

```
root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml
root@HOST1 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
root@HOST2 $ kubeadm init phase etcd local --config=/home/ubuntu/kubeadmcfg.yaml
```

h Validation finale

```
docker run --rm -it \
--net host \
-v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd:${ETCD_TAG} etcdctl \
--cert-file /etc/kubernetes/pki/etcd/peer.crt \
--key-file /etc/kubernetes/pki/etcd/peer.key \
--ca-file /etc/kubernetes/pki/etcd/ca.crt \
--endpoints https://${HOST0}:2379 cluster-health
...
cluster is healthy
```

L'AC créée pour etcd sera utilisée plus tard lors de l'approvisionnement des nœuds maîtres.

29.3 Configurer kubeadm pour les master

La configuration de kubeadm permet de personnaliser les certificats et manifestes nécessaires au déploiement de Kubernetes. Il devrait inclure des informations sur les api, les certs etcd, le jeton bootstrap et les noms d'hôte/adresses IP qui devraient être inclus dans les certificats.

Avant de configurer la configuration, générez un token.

```
kubeadm token generate
```

a Exemple de fichier de configuration

```
apiVersion: kubeadm.k8s.io/v1alpha2
kind: MasterConfiguration
kubernetesVersion: v1.11.3
api:
  controlPlaneEndpoint: "10.0.0.200:6443"
etcd:
  external:
    endpoints:
    - https://10.0.0.6:2379
    - https://10.0.0.7:2379
    - https://10.0.0.8:2379
    caFile: /etc/kubernetes/pki/etcd/ca.crt
    certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt
    keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key
apiServerExtraArgs:
  apiserver-count: "3"
apiServerCertSANs:
- kubernetes.default
- kubernetes.default.svc.cluster.local
- 10.0.0.200
- 10.0.0.50
- 10.0.0.51
- 10.0.0.52
- master-01-hostname
- master-02-hostname
- master-03-hostname
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: <your token from the step above>
  ttl: '0'
  usages:
  - signing
  - authentication
```

Les certifications etcd doivent avoir été générées lors de la configuration du cluster etcd (ci-haut). Assurez-vous d'inclure les SAN supplémentaires pour votre certificat de serveur API. Il doit inclure toutes les adresses master **ET** l'adresse du loadbalancer ! Notifier également la version api de kubeadm. Enregistrer ce fichier sous le nom kubeadmcfg.yaml

Traduit avec www.DeepL.com/Translator

30 Annexe

30.1 Etcd yaml

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: backup
  namespace: kube-system
spec:
  # activeDeadlineSeconds: 100
  schedule: "*/3 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            # Same image as in /etc/kubernetes/manifests/etcd.yaml
            image: k8s.gcr.io/etcd-amd64:3.1.12
            env:
            - name: ETCDCTL_API
              value: "3"
            command: ["/bin/sh"]
            args: ["-c", "etcdctl --endpoints=https://127.0.0.1:2379
--cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/healthcheck-
client.crt --key=/etc/kubernetes/pki/etcd/healthcheck-client.key snapshot save
/backup/etcd-snapshot-$(date +%Y-%m-%d_%H:%M:%S_%Z).db"]
            volumeMounts:
            - mountPath: /etc/kubernetes/pki/etcd
              name: etcd-certs
              readOnly: true
            - mountPath: /backup
              name: backup
          restartPolicy: OnFailure
          nodeSelector:
            kubernetes.io/hostname: kube-master
          tolerations:
          - effect: NoSchedule
            operator: Exists
          hostNetwork: true
          volumes:
          - name: etcd-certs
            hostPath:
              path: /etc/kubernetes/pki/etcd
              type: DirectoryOrCreate
          - name: backup
            flexVolume:
              driver: "fstab/cifs"
              fsType: "cifs"
              secretRef:
                name: "backup-volume-credentials"
              options:
                networkPath: "//my-server.com/backup"
```

```
mountOptions: "dir_mode=0755,file_mode=0644,noperm"
```