Version - **2020.03**

Dernière mise-à-jour : 2020/12/31 10:14

DOF305 - Sécurisation de Kubernetes

Contenu du Module

LAB #1 - Role Based Acces Control et Certificats TLS

1.1 - Présentation

Un objet Kubernetes est soit lié à un Namespace soit non-lié à un Namespace.

Kubernetes utilise l'API **rbac.authorization.k8s.io** pour gérer les autorisations. Les acteurs jouant un rôle dans cette API sont :

- **Namespaces**,
  - peuvent être considérées comme des clusters virtuels,
  - permettent l'isolation et la segmentation logique,
  - permettent le regroupement d'utilisateurs, de rôles et de ressources,
  - sont utilisés avec des applications, des clients, des projets ou des équipes.
- **Subjects**,
  - *Regular Users* - permettent la gestion des accès autorisés depuis l'extérieur du cluster que cela soit par un utilisateur physique ou sous une autre forme. La gestion des utilisateurs est la responsabilité de l'Administrateur du cluster,
  - *ServiceAccounts* - permettent la mise en place de permissions au niveau des entités logiciels. Kubernetes crée un certain nombre de serviceAccounts automatiquement mais l'Administrateur peut en créer d'autres. Chaque pod a un serviceAccount qui gère les privilèges accordés au processus et aux conteneurs du pod,
  - *User Groups* - Kubernetes regroupe des utilisateurs en utilisant des propriétés communes telles le préfixe d'un serviceAccount ou le champ de l'organisation dans un certificat. Il est ensuite possible d'accorder des privilèges de type RBAC aux groupes ainsi créés.
- **Resources**,
  - ce sont des entités auxquelles auront accès les Subjects,
  - une ressource est une entité telle un pod, un deployment ou des sous-ressources telles les journaux d'un pod,
  - le Pod Security Policy (PSP) est aussi considéré comme une ressource.
- **Roles** et **ClusterRoles**,
  - *Roles* - permettent de définir des règles représentant un jeu de permissions, telles GET WATCH LIST CREATE UPDATE PATCH et DELETE, qui peuvent être utilisées avec des ressources dans un Namespace,
    - On ajoute des permissions, on ne les retire pas. Il n'y a pas donc des règles de type **deny**.
  - *ClusterRoles* - n'est pas lié à un Namespace. Un ClusterRole est utilisé pour :
    - définir des permissions pour des ressources à être utilisées dans un Namespace

    - définir des permissions pour des ressources à être utilisées dans tous les Namespaces
    - définir des permissions pour des ressources du cluster.

- Un exemple d'un Role pour accorder les permissions dans le Namespace default est :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

- Un example d'un ClusterRole pour accorder des permissions de lecture des secrets dans un Namespace spécifique ou dans tous les Namespaces est :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

- **RoleBindings** et **ClusterRoleBindings**,
    - permettent d'accorder des permissions définies dans des Roles ou ClusterRoles à des Subjects,
    - **RoleBindings** sont spécifiques à un NameSpace,
    - **ClusterRoleBindings** s'appliquent au niveau du Cluster.

1.2 - Le Fichier /etc/kubernetes/manifests/kube-apiserver.yaml

Connectez-vous au noeud **kind-control-plane** en utilisant la commande **docker** :

```
root@debian10:~# docker exec -it kind-control-plane /bin/bash
root@kind-control-plane:/#
```

L'utilisation de RBAC est définie par la valeur de la directive **–authorization-mode** dans le fichier **/etc/kubernetes/manifests/kube-apiserver.yaml** :

```
root@kind-control-plane:/# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 172.18.0.5:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=172.18.0.5
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

```
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --runtime-config=
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-cluster-ip-range=10.96.0.0/16
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
--More--
[q] <-----------------------------------------------------------Appuyez sur la touche **q**
root@kind-control-plane:/# exit
exit
```

1.3 - Création d'un serviceAccount

Il est préférable de créer un serviceAccount par service. Ceci permet une configuration plus fine de la sécurité concernant le service. Si un serviceAccount n'est pas spécifié lors de la création des pods, ces pods se verront attribués le serviceAccount par défaut du Namespace.

Imaginons que vous souhaitez que votre application interagisse avec l'API de Kubernetes afin d'obtenir des informations sur les pods dans un Namespace. le serviceAccount par défaut dasn le Namespace **default** ne peut pas accomplir cette tâche :

```
root@debian10:~# kubectl auth can-i list pods -n default --as=system:serviceaccount:default:default
no
```

Important : le format de la valeur de l'option **–as** est **system:serviceaccount:namespace:Nom_du_serviceaccount**.

Créez maintenant le fichier **flask.yaml** :

```
root@debian10:~# vi flask.yaml
root@debian10:~# cat flask.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: flask
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: flask-backend
  namespace: flask
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: flask-backend-role
  namespace: flask
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: flask-backend-role-binding
  namespace: flask
subjects:
  - kind: ServiceAccount
    name: flask-backend
    namespace: flask
roleRef:
  kind: Role
  name: flask-backend-role
  apiGroup: rbac.authorization.k8s.io
```

Ce fichier crée :

- un Namespace appelé **flask**,
- un serviceAccount appelé **flask-backend** pour le Namespace **flask**,
- un Role appelé **flask-backend-role** qui accorde les permissions **get**, **watch** et **list** sur les pods dans le Namespace **flask**,
- un RoleBinding appelé **flask-backend-role-binding** qui accorde les permissions définies dans le Role **flask-backend-role** au Subject de type serviceAccount appelé **flask-backend**.

> **Important** : apiGroups: [“”] - “” indique le groupe api core ou legacy. Ce groupe se trouve au chemin REST /api/v1. Ce groupe n'est jamais spécifié dans un champs apiVersion, d'où la raison pour laquelle on écrit apiVersion: v1 et non apiVersion api/v1.

Appliquez le fichier :

```
root@debian10:~# kubectl create -f flask.yaml
namespace/flask created
serviceaccount/flask-backend created
role.rbac.authorization.k8s.io/flask-backend-role created
```

```
rolebinding.rbac.authorization.k8s.io/flask-backend-role-binding created
```

Créez maintenant le fichier **deployment_flask.yaml** qui crée des pods qui utiliseront le serviceAccount appelé **flask-backend** :

```
root@debian10:~# vi deployment_flask.yaml
root@debian10:~# cat deployment_flask.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  namespace: flask
  labels:
    app: myapp
    type: front-end
spec:
  template:

    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      serviceAccount: flask-backend
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

Exécutez kubectl :

```
root@debian10:~# kubectl create -f deployment_flask.yaml
deployment.apps/myapp-deployment created
```

Vérifiez la présence du deployment :

```
root@debian10:~# kubectl get deployment -n flask
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
myapp-deployment   3/3     3            3           25s
```

Vérifiez qu'un token d'accès à l'API a été créé :

```
root@debian10:~# kubectl get secrets -n flask
NAME                      TYPE                                  DATA   AGE
default-token-k2s9s       kubernetes.io/service-account-token   3      3m8s
flask-backend-token-b2n9r   kubernetes.io/service-account-token   3      3m8s
```

Vérifiez maintenant que le serviceAccount **flask-backend** peut lister les pods dans le Namespace **flask** :

```
root@debian10:~# kubectl auth can-i list pods -n flask --as=system:serviceaccount:flask:flask-backend
yes
```

Notez cependant que le serviceAccount **flask-backend** n'a pas la permission **create** dans le Namespace **flask** :

```
root@debian10:~# kubectl auth can-i create pods -n flask --as=system:serviceaccount:flask:flask-backend
no
```

et que le serviceAccount **flask-backend** n'a pas la permission **list** dans le Namespace **default** :

```
root@debian10:~# kubectl auth can-i list pods -n default --as=system:serviceaccount:flask:flask-backend
no
```

1.4 - Création d'un Utilisateur

Connectez-vous au noeud **kind-control-plane** en utilisant la commande **docker** :

```
root@debian10:~# docker exec -it kind-control-plane /bin/bash
root@kind-control-plane:/#
```

Configurez l'accès au cluster :

```
root@kind-control-plane:/# mkdir -p $HOME/.kube
root@kind-control-plane:/# cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@kind-control-plane:/# chown $(id -u):$(id -g) $HOME/.kube/config
```

Les utilisateurs font partis du contexte de configuration qui définit le nom du cluster et le nom du Namespace :

```
root@kind-control-plane:/# kubectl config get-contexts
CURRENT   NAME                   CLUSTER   AUTHINFO           NAMESPACE
*         kubernetes-admin@kind   kind      kubernetes-admin
```

Important : Un contexte est un élément qui regroupe les paramètres d'accès sous un nom. Les paramètres d'accès sont au nombre de trois, à savoir le cluster, le namespace et l'utilisateur. La commande kubectl utilise les paramètres du contexte courant pour communiquer avec le cluster.

Consultez le contexte courant :

```
root@kind-control-plane:/# kubectl config view
apiVersion: v1
clusters:
```

```
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://kind-control-plane:6443
  name: kind
contexts:
- context:
    cluster: kind
    user: kubernetes-admin
  name: kubernetes-admin@kind
current-context: kubernetes-admin@kind
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

Important : Le mot **REDACTED** indique que les valeurs sont cachées pour des raisons de sécurité.

Pour créer un nouveau utilisateur il faut commencer par créer une clef privée pour l'utilisateur :

```
root@kind-control-plane:/# openssl genrsa -out trainee.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................+++++
.......+++++
e is 65537 (0x010001)
```

Créez maintenant un CSR :

```
root@kind-control-plane:/# openssl req -new -key trainee.key -out trainee.csr -subj "/CN=trainee/O=examplegroup"
```

Important : Notez que Kubernetes utilisera la valeur de la clef de l'organisation pour le regroupement des utilisateurs.

Le CSR doit être signé par le CA racine de Kubernetes :

```
root@kind-control-plane:/# ls -l /etc/kubernetes/pki/ca.*
-rw-r--r-- 1 root root 1066 Dec 13 12:34 /etc/kubernetes/pki/ca.crt
-rw------- 1 root root 1675 Dec 13 12:34 /etc/kubernetes/pki/ca.key
```

Signez donc le CSR :

```
root@kind-control-plane:/# openssl x509 -req -in trainee.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out trainee.crt
Signature ok
subject=CN = trainee, O = examplegroup
Getting CA Private Key
```

Visualisez le certificat de trainee :

```
root@kind-control-plane:/# openssl x509 -in trainee.crt -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            65:7a:9f:e1:5d:bd:48:27:b3:50:c9:9f:60:c8:04:85:4e:85:7b:02
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = kubernetes
        Validity
            Not Before: Dec 15 13:36:03 2020 GMT
```

```
            Not After : Jan 14 13:36:03 2021 GMT
        Subject: CN = trainee, O = examplegroup
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:b1:41:77:ae:dd:43:01:ee:73:57:73:46:f4:64:
                    84:52:49:7d:75:0c:26:e1:da:4e:46:e0:ba:af:84:
                    7d:05:51:db:da:51:99:26:37:b1:6b:a0:29:c7:fa:
                    c8:0a:ca:ab:97:34:7d:79:a8:ea:00:75:9c:22:fd:
                    40:1e:dd:85:72:29:dd:18:c1:9e:9c:b6:e7:54:a1:
                    65:9a:08:e5:e5:22:1a:cc:71:33:6e:71:d2:2b:1d:
                    1b:ed:01:06:5e:42:33:db:97:a0:9b:c7:59:b2:ca:
                    0f:37:f2:a7:23:a9:6e:cd:63:fd:62:ac:b9:c6:aa:
                    47:e7:4c:7f:8b:b5:03:cd:8d:09:b6:98:c4:79:0e:
                    97:b5:7b:8c:69:0b:10:a3:05:1d:8f:a6:dd:42:dc:
                    60:76:08:1d:a2:9f:76:42:fe:77:6b:4e:d0:dc:a8:
                    54:ab:e2:07:9e:0d:d0:1c:18:f5:c0:ca:3f:d9:32:
                    5c:45:fd:e5:b3:94:bc:04:0e:49:fa:e6:2e:c1:e1:
                    f2:b4:88:38:9b:d5:b8:27:d3:6f:8b:80:ac:bb:59:
                    17:20:9c:6a:71:72:5a:3e:c9:05:07:e2:6d:33:e2:
                    2e:78:58:85:46:89:ca:74:3e:d1:3a:a5:6c:1b:cb:
                    f0:84:62:8b:44:9c:bb:1b:ab:d8:81:77:37:c0:c9:
                    27:15
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         33:14:25:3c:51:ac:63:0b:a6:6f:1e:49:aa:3b:9c:e2:56:47:
         ea:23:d7:2f:e7:b0:4b:c0:6f:a7:51:19:2d:2e:ad:5e:ef:be:
         07:3e:96:f5:db:6c:64:6d:83:0d:81:51:04:43:0c:ce:0b:09:
         de:ed:d3:c9:b5:85:b7:09:b9:20:67:31:4d:e1:09:7c:ec:e3:
         ef:b4:10:ff:7a:fe:8e:20:6c:0f:d5:b5:1f:73:f2:a9:d9:ec:
         84:eb:c2:ab:1d:0e:27:ab:a8:b5:11:a6:d6:55:98:e6:90:2d:
         25:e4:85:f1:70:04:e3:8e:9a:a8:bf:aa:a7:6a:ea:9f:5a:72:
         f1:20:7d:f5:a6:1d:0b:0e:bd:ba:d8:93:a6:a1:62:99:01:5d:
```

```
        bf:70:60:25:cd:cb:a8:62:39:e7:81:5c:79:da:b4:23:4e:54:
        a5:f4:cd:7a:58:13:a7:de:da:02:80:d9:9f:2b:6c:00:c1:43:
        a8:a9:f8:de:1f:fa:ff:a8:12:a0:ea:e8:1c:aa:2f:99:ee:9c:
        9d:09:63:79:f8:7f:2d:12:67:aa:dd:71:70:c1:b6:19:3e:fe:
        1e:74:4e:71:41:01:49:9e:51:38:d2:00:eb:2e:b8:cf:4f:0d:
        19:9b:2b:0c:57:d7:dc:4d:23:d0:17:27:42:39:bf:ec:7c:e9:
        22:ac:23:2e
-----BEGIN CERTIFICATE-----
MIICxTCCAa0CFGV6n+FdvUgns1DJn2DIBIVOhXsCMA0GCSqGSIb3DQEBCwUAMBUx
EzARBgNVBAMTCmt1YmVybmV0ZXMwHhcNMjAxMjE1MTMzNjAzWhcNMjEwMTE0MTMz
NjAzWjApMRAwDgYDVQQDDAd0cmFpbmVlMRUwEwYDVQQKDAxleGFtcGxlZ3JvdXAw
ggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCxQXeu3UMB7nNXc0b0ZIRS
SX11DCbh2k5G4LqvhH0FUdvaUZkmN7FroCnH+sgKyquXNH15qOoAdZwi/UAe3YVy
Kd0YwZ6ctudUoWWaCOXlIhrMcTNucdIrHRvtAQZeQjPbl6Cbx1myyg838qcjqW7N
Y/1irLnGqkfnTH+LtQPNjQm2mMR5Dpe1e4xpCxCjBR2Ppt1C3GB2CB2in3ZC/ndr
TtDcqFSr4geeDdAcGPXAyj/ZMlxF/eWzlLwEDkn65i7B4fK0iDib1bgn02+LgKy7
WRcgnGpxclo+yQUH4m0z4i54WIVGicp0PtE6pWwby/CEYotEnLsbq9iBdzfAyScV
AgMBAAEwDQYJKoZIhvcNAQELBQADggEBADMUJTxRrGMLpm8eSao7nOJWR+oj1y/n
sEvAb6dRGS0urV7vvgc+lvXbbGRtgw2BUQRDDM4LCd7t08m1hbcJuSBnMU3hCXzs
4++0EP96/o4gbA/VtR9z8qnZ7ITrwqsdDierqLURptZVmOaQLSXkhfFwBOOOmqi/
qqdq6p9acvEgffWmHQsOvbrYk6ahYpkBXb9wYCXNy6hiOeeBXHnatCNOVKX0zXpY
E6fe2gKA2Z8rbADBQ6ip+N4f+v+oEqDq6ByqL5nunJ0JY3n4fy0SZ6rdcXDBthk+
/h50TnFBAUmeUTjSAOsuuM9PDRmbKwxX19xNI9AXJ0I5v+x86SKsIy4=
-----END CERTIFICATE-----
```

Créez un deuxième utilisateur dans la même Organisation :

```
root@kind-control-plane:/# openssl genrsa -out stagiaire.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
............+++++
.....................................................................................+++++
e is 65537 (0x010001)

root@kind-control-plane:/# openssl req -new -key stagiaire.key -out stagiaire.csr -subj
```

```
"/CN=stagiaire/O=examplegroup"

root@kind-control-plane:/# openssl x509 -req -in stagiaire.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out stagiaire.crt
Signature ok
subject=CN = stagiaire, O = examplegroup
Getting CA Private Key
```

Créez maintenant le contexte **trainee** :

```
root@kind-control-plane:/# kubectl config set-credentials trainee --client-certificate=trainee.crt --client-
key=trainee.key
User "trainee" set.
root@kind-control-plane:/# kubectl config set-context trainee@kubernetes --cluster=kubernetes --user=trainee
Context "trainee@kubernetes" created.
```

Vérifiez que le contexte soit présent :

```
root@kind-control-plane:/# kubectl config get-contexts
CURRENT   NAME                   CLUSTER      AUTHINFO           NAMESPACE
*         kubernetes-admin@kind   kind         kubernetes-admin
          trainee@kubernetes      kubernetes   trainee
```

Utilisez le contexte de trainee :

```
root@kind-control-plane:/# kubectl config use-context trainee@kubernetes
Switched to context "trainee@kubernetes".
```

Vérifiez que vous utilisez le contexte de trainee :

```
root@kind-control-plane:/# kubectl config get-contexts
CURRENT   NAME                   CLUSTER      AUTHINFO           NAMESPACE
          kubernetes-admin@kind   kind         kubernetes-admin
```

```
*         trainee@kubernetes      kubernetes   trainee
```

Revenez au contexte de **kubernetes-admin@kind** :

```
root@kind-control-plane:/# kubectl config use-context kubernetes-admin@kind
Switched to context "kubernetes-admin@kind".
root@kind-control-plane:/# exit
exit
root@debian10:~#
```

1.5 - Certificats TLS

Par défaut la communication entre kubectl et l'API Kubernetes est cryptée. Les certificats se trouvent dans le répertoire **/var/lib/kubelet/pki/** de chaque noeud :

```
root@kind-control-plane:/# ls -l /var/lib/kubelet/pki/
total 12
-rw------- 1 root root 2818 Dec 13 12:34 kubelet-client-2020-12-13-12-34-39.pem
lrwxrwxrwx 1 root root   59 Dec 13 12:34 kubelet-client-current.pem -> /var/lib/kubelet/pki/kubelet-
client-2020-12-13-12-34-39.pem
-rw-r--r-- 1 root root 2319 Dec 13 12:34 kubelet.crt
-rw------- 1 root root 1679 Dec 13 12:34 kubelet.key
root@kind-control-plane:/# exit
root@debian10:~# docker exec -it kind-worker2 /bin/bash
root@kind-worker2:/# ls -l /var/lib/kubelet/pki/
total 12
-rw------- 1 root root 1118 Dec 13 12:35 kubelet-client-2020-12-13-12-35-43.pem
lrwxrwxrwx 1 root root   59 Dec 13 12:35 kubelet-client-current.pem -> /var/lib/kubelet/pki/kubelet-
client-2020-12-13-12-35-43.pem
-rw-r--r-- 1 root root 2279 Dec 13 12:35 kubelet.crt
-rw------- 1 root root 1679 Dec 13 12:35 kubelet.key
root@kind-worker2:/# exit
exit
```

```
root@debian10:~# docker exec -it kind-worker3 /bin/bash
root@kind-worker3:/# ls -l /var/lib/kubelet/pki/
total 12
-rw------- 1 root root 1118 Dec 13 12:35 kubelet-client-2020-12-13-12-35-45.pem
lrwxrwxrwx 1 root root   59 Dec 13 12:35 kubelet-client-current.pem -> /var/lib/kubelet/pki/kubelet-
client-2020-12-13-12-35-45.pem
-rw-r--r-- 1 root root 2279 Dec 13 12:35 kubelet.crt
-rw------- 1 root root 1675 Dec 13 12:35 kubelet.key
root@kind-worker3:/# exit
exit
```

Important : Par défaut les certificats de kubelet expirent au bout d'un an.

LAB #2 - Implémentation de la Sécurité au niveau des Pods

2.1 - Présentation

Un **Admission Controller** est un morceau de code qui intercepte les requêtes à destination de l'API de Kubernetes. L'utilisation des Admission Controllers est définie part la directive **–admission-control** du fichier **/etc/kubernetes/manifests/kube-apiserver.yaml**, par exemple :

```
--admission-control=Initializers, NamespaceLifecycle, LimitRanger, ServiceAccount, PersistentVolumeLabel,
DefaultStorageClass, DefaultTolerationSeconds, NodeRestriction, ResourceQuota
```

Les Admission Controllers les plus importants en termes de sécurité sont :

- **DenyEscalatingExec**,
  - interdit l'exécution des commandes avec un *escalated container* dans un pod priviligié. Les commandes concernées sont **exec** et **attach**. Un *escalated container* dans un pod priviligié n'est pas **isolé** et permet donc l'accès à l'hôte.
- **NodeRestriction**,

    - limite les objets d'un nœud et d'un pod que kubectl est capable de modifier,
- **PodSecurityPolicy**,
    - agit lors de la création ou de la modification d'un pod pour décider si celui-ci est admis au cluster en fonction du Contexte de Sécurité et les policies applicables,
- **ValidatingAdmissionWebhooks**,
    - permet d'appeler un service externe qui implémente une politique de sécurité, tel que **Grafeas**.

2.2 - Kubernetes Security Context

La configuration du Contexte de Sécurité se fait du pod ou du conteneur. Voici quelques exemples.

ReadOnlyRootFilesystem

Créez le fichier **readonly.yaml** :

```
root@debian10:~# vi readonly.yaml
root@debian10:~# cat readonly.yaml
apiVersion: v1
kind: Pod
metadata:
  name: flask-ro
  namespace: default
spec:
  containers:
  - image: mateobur/flask
    name: flask-ro
    securityContext:
      readOnlyRootFilesystem: true
```

Exécutez kubectl :

```
root@kubemaster:~# kubectl create -f readonly.yaml
pod/flask-ro created
```

Vérifiez que le pod est en état de **READY** :

```
root@debian10:~# kubectl create -f readonly.yaml
pod/flask-ro created

root@debian10:~# kubectl get pods
NAME                                  READY   STATUS              RESTARTS   AGE
flask-ro                              0/1     ContainerCreating   0          11s
postgres-deployment-746bc85b8-8lw6c   1/1     Running             1          4h23m
redis-deployment-64cff75679-8zqr8     1/1     Running             1          4h23m
result-app-deployment-7cdc94dfcd-nddsh   1/1     Running             1          4h23m
result-app-deployment-7cdc94dfcd-ntbdj   1/1     Running             0          3h15m
result-app-deployment-7cdc94dfcd-wsm2d   1/1     Running             0          3h15m
voting-app-deployment-678c67fc7-59q7z    1/1     Running             0          3h15m
voting-app-deployment-678c67fc7-sgczf    1/1     Running             0          3h15m
voting-app-deployment-678c67fc7-zcs6c    1/1     Running             1          4h23m
worker-app-deployment-767d5b67ff-sgj2x   1/1     Running             2          4h23m

root@debian10:~# kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
flask-ro                              1/1     Running   0          29s
postgres-deployment-746bc85b8-8lw6c   1/1     Running   1          4h24m
redis-deployment-64cff75679-8zqr8     1/1     Running   1          4h24m
result-app-deployment-7cdc94dfcd-nddsh   1/1     Running   1          4h24m
result-app-deployment-7cdc94dfcd-ntbdj   1/1     Running   0          3h15m
result-app-deployment-7cdc94dfcd-wsm2d   1/1     Running   0          3h15m
voting-app-deployment-678c67fc7-59q7z    1/1     Running   0          3h15m
voting-app-deployment-678c67fc7-sgczf    1/1     Running   0          3h15m
voting-app-deployment-678c67fc7-zcs6c    1/1     Running   1          4h23m
worker-app-deployment-767d5b67ff-sgj2x   1/1     Running   2          4h23m
```

Connectez-vous au conteneur :

```
root@debian10:~# kubectl exec -it flask-ro -- bash
root@flask-ro:/#
```

Notez que le système est en lecture seule :

```
root@flask-ro:/# mount | grep "/ "
overlay on / type overlay
(ro,relatime,lowerdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/57/fs:/var/lib/contain
erd/io.containerd.snapshotter.v1.overlayfs/snapshots/56/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overl
ayfs/snapshots/55/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/54/fs,upperdir=/var/lib
/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/58/fs,workdir=/var/lib/containerd/io.containerd.snap
shotter.v1.overlayfs/snapshots/58/work)

root@flask-ro:/# touch test
touch: cannot touch 'test': Read-only file system

root@flask-ro:/# exit
exit
command terminated with exit code 1
```

drop

Créez le fichier **drop.yaml** :

```
root@debian10:~# vi drop.yaml
root@debian10:~# cat drop.yaml
apiVersion: v1
kind: Pod
metadata:
  name: flask-cap
```

```
  namespace: default
spec:
  containers:
  - image: mateobur/flask
    name: flask-cap
    securityContext:
      capabilities:
        drop:
          - NET_RAW
          - CHOWN
```

Exécutez kubectl :

```
root@debian10:~# kubectl create -f drop.yaml
pod/flask-cap created
```

Vérifiez que le pod est en état de **READY** :

```
root@debian10:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-cap                           1/1     Running   0          23s
flask-ro                            1/1     Running   0          5m42s
postgres-deployment-746bc85b8-8lw6c      1/1     Running   1          4h29m
redis-deployment-64cff75679-8zqr8        1/1     Running   1          4h29m
result-app-deployment-7cdc94dfcd-nddsh   1/1     Running   1          4h29m
result-app-deployment-7cdc94dfcd-ntbdj   1/1     Running   0          3h20m
result-app-deployment-7cdc94dfcd-wsm2d   1/1     Running   0          3h20m
voting-app-deployment-678c67fc7-59q7z    1/1     Running   0          3h21m
voting-app-deployment-678c67fc7-sgczf    1/1     Running   0          3h21m
voting-app-deployment-678c67fc7-zcs6c    1/1     Running   1          4h29m
worker-app-deployment-767d5b67ff-sgj2x   1/1     Running   2          4h29m
```

Connectez-vous au conteneur :

```
root@debian10:~# kubectl exec -it flask-cap -- bash
root@flask-cap:/#
```

Notez la mise en place des restrictions :

```
root@flask-cap:/# ping 8.8.8.8
ping: Lacking privilege for raw socket.

root@flask-cap:/# chown daemon /tmp
chown: changing ownership of '/tmp': Operation not permitted

root@flask-cap:/# exit
exit
command terminated with exit code 1
```

HERE

2.3 - Kubernetes Pod Security Policy

Créez le fichier **psp.yaml** :

```
root@kubemaster:~# vi psp.yaml
root@kubemaster:~# cat psp.yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: true
  seLinux:
    rule: RunAsAny
  supplementalGroups:
```

```
    rule: RunAsAny
  runAsUser:
    rule: 'MustRunAs'
    ranges:
      - min: 1
        max: 65535
  fsGroup:
    rule: 'MustRunAs'
    ranges:
      - min: 1
        max: 65535
  volumes:
  - '*'
```

Exécutez kubectl :

```
root@kubemaster:~# kubectl create -f psp.yaml
podsecuritypolicy.policy/example created
```

Consultez la présence de la **Pod Security Policy** :

```
root@kubemaster:~# kubectl get psp
NAME      PRIV   CAPS   SELINUX    RUNASUSER   FSGROUP     SUPGROUP   READONLYROOTFS   VOLUMES
example   true          RunAsAny   MustRunAs   MustRunAs   RunAsAny   false            *
```

La **Pod Security Policy** créée empêche l'exécution d'un pod en utilisant l'utilisateur et le groupe **root**.

2.4 - Kubernetes Network Policies

Installez l'application exemple **Guestbook** de Kubernetes :

```
root@kubemaster:~# kubectl create -f
```

```
https://raw.githubusercontent.com/fabric8io/kansible/master/vendor/k8s.io/kubernetes/examples/guestbook/all-in-on
e/guestbook-all-in-one.yaml
service/redis-master created
replicationcontroller/redis-master created
service/redis-slave created
replicationcontroller/redis-slave created
service/frontend created
replicationcontroller/frontend created
```

Cette application crée des pods de type *backend* et *frontend* :

```
root@kubemaster:~# kubectl describe pod redis-master-8rczl | grep tier
                tier=backend
root@kubemaster:~# kubectl describe pod frontend-762mw | grep tier
                tier=frontend
```

Créez le fichier **guestbook-network-policy.yaml** qui empêchera la communication d'un pod backend vers un pod frontend :

```
root@kubemaster:~# vi guestbook-network-policy.yaml
root@kubemaster:~# cat guestbook-network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-backend-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
    - Egress
  egress:
    - to:
       - podSelector:
```

```
      matchLabels:
        tier: backend
```

Exécutez kubectl :

```
root@kubemaster:~# kubectl create -f guestbook-network-policy.yaml
networkpolicy.networking.k8s.io/deny-backend-egress created
```

Attendez que tous les pods soient dans un état de **READY** :

```
root@kubemaster:~# kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE    IP               NODE        NOMINATED NODE   READINESS GATES
flask-cap                             1/1     Running   1          139m   192.168.205.242   kubenode1   <none>           <none>
flask-ro                              1/1     Running   0          65m    192.168.35.141    kubenode2   <none>           <none>
frontend-762mw                        1/1     Running   0          17m    192.168.205.250   kubenode1   <none>           <none>
frontend-lhw8b                        1/1     Running   0          17m    192.168.35.143    kubenode2   <none>           <none>
frontend-n75vs                        1/1     Running   0          17m    192.168.205.252   kubenode1   <none>           <none>
postgres-deployment-5b8bd66778-j99zz  1/1     Running   8          4d3h   192.168.35.138    kubenode2   <none>           <none>
redis-deployment-67d4c466c4-9wzfn     1/1     Running   8          4d3h   192.168.205.246   kubenode1   <none>           <none>
redis-master-8rczl                    1/1     Running   0          17m    192.168.205.249   kubenode1   <none>           <none>
redis-slave-c8jzv                     1/1     Running   0          17m    192.168.35.142    kubenode2   <none>           <none>
redis-slave-fjrjm                     1/1     Running   0          17m    192.168.205.251   kubenode1   <none>           <none>
result-app-deployment-b8f9dc967-nzbgd 1/1     Running   8          4d3h   192.168.205.245   kubenode1   <none>
```

```
<none>
result-app-deployment-b8f9dc967-r84k6    1/1     Running   8          4d     192.168.35.135    kubenode2   <none>
<none>
result-app-deployment-b8f9dc967-zbsk2    1/1     Running   8          4d     192.168.35.137    kubenode2   <none>
<none>
voting-app-deployment-669dccccfb-jpn6h   1/1     Running   8          4d3h   192.168.35.136    kubenode2   <none>
<none>
voting-app-deployment-669dccccfb-ktd7d   1/1     Running   8          4d     192.168.35.140    kubenode2   <none>
<none>
voting-app-deployment-669dccccfb-x868p   1/1     Running   8          4d     192.168.205.243   kubenode1   <none>
<none>
worker-app-deployment-559f7749b6-jh86r   1/1     Running   21         4d3h   192.168.205.248   kubenode1   <none>
<none>
```

Connectez-vous au pod **redis-master** :

```
root@kubemaster:~# kubectl exec -it redis-master-8rczl bash
[ root@redis-master-8rczl:/data ]$
```

Essayez de contacter un pod du même **tier** :

```
[ root@redis-master-8rczl:/data ]$ ping -c 4 192.168.35.142
PING 192.168.35.142 (192.168.35.142) 56(84) bytes of data.
64 bytes from 192.168.35.142: icmp_seq=1 ttl=62 time=0.402 ms
64 bytes from 192.168.35.142: icmp_seq=2 ttl=62 time=0.301 ms
64 bytes from 192.168.35.142: icmp_seq=3 ttl=62 time=0.291 ms
64 bytes from 192.168.35.142: icmp_seq=4 ttl=62 time=0.395 ms

--- 192.168.35.142 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.291/0.347/0.402/0.053 ms
```

Essayez maintenant de contacter un pod d'un **tier** différent :

```
[ root@redis-master-8rczl:/data ]$ ping -c 4 192.168.205.250
PING 192.168.205.250 (192.168.205.250) 56(84) bytes of data.

--- 192.168.205.250 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3017ms
```

Déconnectez-vous du pod **redis-master** et connectez-vous à un pod **frontend** :

```
root@kubemaster:~# kubectl exec -it frontend-762mw bash
root@frontend-762mw:/var/www/html#
```

Essayez de contacter un pod du même **tier** :

```
root@frontend-762mw:/var/www/html# ping -c 4 192.168.35.143
PING 192.168.35.143 (192.168.35.143): 56 data bytes
64 bytes from 192.168.35.143: icmp_seq=0 ttl=62 time=0.476 ms
64 bytes from 192.168.35.143: icmp_seq=1 ttl=62 time=0.263 ms
64 bytes from 192.168.35.143: icmp_seq=2 ttl=62 time=0.231 ms
64 bytes from 192.168.35.143: icmp_seq=3 ttl=62 time=0.289 ms
--- 192.168.35.143 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.231/0.315/0.476/0.095 ms
```

Essayez maintenant de contacter un pod d'un **tier** différent :

```
root@frontend-762mw:/var/www/html# ping -c 4 192.168.205.249
PING 192.168.205.249 (192.168.205.249): 56 data bytes
64 bytes from 192.168.205.249: icmp_seq=0 ttl=63 time=0.454 ms
64 bytes from 192.168.205.249: icmp_seq=1 ttl=63 time=0.052 ms
64 bytes from 192.168.205.249: icmp_seq=2 ttl=63 time=0.069 ms
64 bytes from 192.168.205.249: icmp_seq=3 ttl=63 time=0.050 ms
--- 192.168.205.249 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
```

```
round-trip min/avg/max/stddev = 0.050/0.156/0.454/0.172 ms
```

2.5 - Kubernetes Resource Allocation Management

Les ressources qui peuvent être limitées au niveau d'un pod sont :

- CPU
- Mémoire
- Stockage local

Créez le fichier **flask-resources.yaml** :

```
root@kubemaster:~# vi flask-resources.yaml
root@kubemaster:~# cat flask-resources.yaml
apiVersion: v1
kind: Pod
metadata:
  name: flask-resources
  namespace: default
spec:
  containers:
  - image: mateobur/flask
    name: flask-resources
    resources:
      requests:
        memory: 512Mi
      limits:
        memory: 700Mi
```

Dans ce fichier on peut constater deux allocations de ressources :

- **requests**,
    - la quantité de mémoire qui doit être libre au moment du scheduling du pod,

- **limits**,
  - la limite de mémoire pour le pod concerné.

Exécutez kubectl :

```
root@kubemaster:~# kubectl create -f flask-resources.yaml
pod/flask-resources created
```

Attendez que le statut du pod soit **READY** :

```
root@kubemaster:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-cap                           1/1     Running   1          161m
flask-resources                     1/1     Running   0          4m47s
flask-ro                            1/1     Running   0          87m
frontend-762mw                      1/1     Running   0          39m
frontend-lhw8b                      1/1     Running   0          39m
frontend-n75vs                      1/1     Running   0          39m
postgres-deployment-5b8bd66778-j99zz    1/1     Running   8          4d3h
redis-deployment-67d4c466c4-9wzfn       1/1     Running   8          4d3h
redis-master-8rczl                  1/1     Running   0          39m
redis-slave-c8jzv                   1/1     Running   0          39m
redis-slave-fjrjm                   1/1     Running   0          39m
result-app-deployment-b8f9dc967-nzbgd   1/1     Running   8          4d3h
result-app-deployment-b8f9dc967-r84k6   1/1     Running   8          4d
result-app-deployment-b8f9dc967-zbsk2   1/1     Running   8          4d
voting-app-deployment-669dccccfb-jpn6h  1/1     Running   8          4d3h
voting-app-deployment-669dccccfb-ktd7d  1/1     Running   8          4d
voting-app-deployment-669dccccfb-x868p  1/1     Running   8          4d
worker-app-deployment-559f7749b6-jh86r  1/1     Running   21         4d3h
```

Connectez-vous au pod :

```
root@kubemaster:~# kubectl exec -it flask-resources bash
```

```
root@flask-resources:/#
```

Installez le paquet **stress** :

```
root@flask-resources:~# apt install stress
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  stress
0 upgraded, 1 newly installed, 0 to remove and 3 not upgraded.
Need to get 18.5 kB of archives.
After this operation, 44.0 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian/ jessie/main stress amd64 1.0.1-1+deb8u1 [18.5 kB]
Fetched 18.5 kB in 5s (3605 B/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package stress.
(Reading database ... 9595 files and directories currently installed.)
Preparing to unpack .../stress_1.0.1-1+deb8u1_amd64.deb ...
Unpacking stress (1.0.1-1+deb8u1) ...
Setting up stress (1.0.1-1+deb8u1) ...
root@flask-resources:~# which stress
/usr/bin/stress
```

Testez la limite mise en place :

```
root@flask-resources:~# stress --cpu 1 --io 1 --vm 2 --vm-bytes 800M
stress: info: [42] dispatching hogs: 1 cpu, 1 io, 2 vm, 0 hdd
stress: FAIL: [42] (416) <-- worker 46 got signal 9
stress: WARN: [42] (418) now reaping child worker processes
stress: FAIL: [42] (452) failed run completed in 1s
```

LAB #3 - Sécuriser les Composants de Kubernetes

3.1 - L'accès à l'API Kubelet

L'accès à l'API Kubelet est sécurisé par défaut. Par contre il convient de vérifier que la configuration ne comporte pas d'erreurs.

Le fichier **/etc/systemd/system/kubelet.service.d/10-kubeadm.conf** démontre l'emplacement du fichier de configuration de kubelet :

```
root@kubemaster:~# cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS
variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user
should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be
sourced from this file.
EnvironmentFile=-/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS
$KUBELET_EXTRA_ARGS
```

La configuration par défaut de kubelet se trouve dans le fichier **/var/lib/kubelet/config.yaml** :

```
root@kubemaster:~# cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
```

```
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
cgroupDriver: cgroupfs
cgroupsPerQOS: true
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
configMapAndSecretChangeDetectionStrategy: Watch
containerLogMaxFiles: 5
containerLogMaxSize: 10Mi
contentType: application/vnd.kubernetes.protobuf
cpuCFSQuota: true
cpuCFSQuotaPeriod: 100ms
cpuManagerPolicy: none
cpuManagerReconcilePeriod: 10s
enableControllerAttachDetach: true
enableDebuggingHandlers: true
enforceNodeAllocatable:
- pods
eventBurst: 10
eventRecordQPS: 5
evictionHard:
  imagefs.available: 15%
  memory.available: 100Mi
```

```
  nodefs.available: 10%
  nodefs.inodesFree: 5%
evictionPressureTransitionPeriod: 5m0s
failSwapOn: true
fileCheckFrequency: 20s
hairpinMode: promiscuous-bridge
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 20s
imageGCHighThresholdPercent: 85
imageGCLowThresholdPercent: 80
imageMinimumGCAge: 2m0s
iptablesDropBit: 15
iptablesMasqueradeBit: 14
kind: KubeletConfiguration
kubeAPIBurst: 10
kubeAPIQPS: 5
makeIPTablesUtilChains: true
maxOpenFiles: 1000000
maxPods: 110
nodeLeaseDurationSeconds: 40
nodeStatusReportFrequency: 1m0s
nodeStatusUpdateFrequency: 10s
oomScoreAdj: -999
podPidsLimit: -1
port: 10250
registryBurst: 10
registryPullQPS: 5
resolvConf: /etc/resolv.conf
rotateCertificates: true
runtimeRequestTimeout: 2m0s
serializeImagePulls: true
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 4h0m0s
```

```
syncFrequency: 1m0s
topologyManagerPolicy: none
volumeStatsAggPeriod: 1m0s
```

En termes de sécurité il est important que la connexion anonyme soit désactivé :

```
...
authentication:
  anonymous:
    enabled: false
...
```

que kubelet utilise un certificat :

```
...
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
...
```

que le mode d'autorisation soit **webhook** qui délègue les décisions d'autorisation à l'API Kubernetes

```
...
authorization:
  mode: Webhook
...
```

3.2 - Sécuriser etcd

La configuration du cluster y compris la configuration du réseau, du stockage, les mots de passe et des données sensibles sont stockées dans **etcd**.

Il est possible de crypter les données sensibles, appelées des **Secrets**, mais cette fonctionnalité n'est pas activée par défaut.

Pour pouvoir procéder au cryptage il faut deux binaires **etcd** et **etcdctl** :

```
root@kubemaster:~# which etcd
root@kubemaster:~# which etcdctl
```

Pour obtenir les deux binaires, téléchargez l'archive **etcd** :

```
root@kubemaster:~# export RELEASE="3.3.13"
root@kubemaster:~# wget
https://github.com/etcd-io/etcd/releases/download/v${RELEASE}/etcd-v${RELEASE}-linux-amd64.tar.gz
```

Désarchivez l'archive et placez-vous dans le répertoire **etcd-v3.3.13-linux-amd64** :

```
root@kubemaster:~# tar xvf etcd-v${RELEASE}-linux-amd64.tar.gz
root@kubemaster:~# cd etcd-v${RELEASE}-linux-amd64
```

Copiez les binaires **etcd** et **etcdctl** vers **usr/local/bin** :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# cp etcd etcdctl /usr/local/bin
root@kubemaster:~/etcd-v3.3.13-linux-amd64# etcd --version
etcd Version: 3.3.13
Git SHA: 98d3084
Go Version: go1.10.8
Go OS/Arch: linux/amd64
```

Vérifiez le chemin de l'emplacement des certificats ainsi que le port d'écoute :

```
root@kubemaster:~# grep -- '--etcd' /etc/kubernetes/manifests/kube-apiserver.yaml
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

Vérifiez que le port **2379** soit à l'écoute :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# ss -tunelp | grep 2379
tcp    LISTEN     0      128    192.168.56.2:2379                 *:*
users:(("etcd",pid=5449,fd=7)) ino:22971 sk:8 <->
tcp    LISTEN     0      128    127.0.0.1:2379                 *:*
users:(("etcd",pid=5449,fd=6)) ino:22970 sk:9 <->
```

Exportez les variables **ETCDCTL_CACERT**, **ETCDCTL_CERT**, **ETCDCTL_KEY** et **ETCDCTL_API** :

```
export ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt
export ETCDCTL_CERT=/etc/kubernetes/pki/apiserver-etcd-client.crt
export ETCDCTL_KEY=/etc/kubernetes/pki/apiserver-etcd-client.key
export ETCDCTL_API=3
```

Créez le fichier **/etc/kubernetes/pki/encryption-config.yaml** afin d'utiliser l'algorithme de cryptage AES avec le mode d'opération **Enchaînement des blocs** :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# vi /etc/kubernetes/pki/encryption-config.yaml
root@kubemaster:~/etcd-v3.3.13-linux-amd64# cat /etc/kubernetes/pki/encryption-config.yaml
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
    providers:
    - aescbc:
        keys:
        - name: key1
          secret: c2VjcmV0IGlzIHNlY3VyZQ==
        - name: key2
          secret: dGhpcyBpcyBwYXNzd29yZA==
    - identity: {}
```

Activez **experimental-encryption-provider-config** dans le fichier **/etc/kubernetes/manifests/kube-apiserver.yaml** :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# vi /etc/kubernetes/manifests/kube-apiserver.yaml
root@kubemaster:~/etcd-v3.3.13-linux-amd64# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.56.2
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
```

```
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-cluster-ip-range=10.96.0.0/12
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    - --experimental-encryption-provider-config=/etc/kubernetes/pki/encryption-config.yaml
```

Re-démarrez le server **k8s_kube-apiserver** pour la prise en compte de la modification du fichier **/etc/kubernetes/manifests/kube-apiserver.yaml** :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# docker stop $(docker ps | grep k8s_kube-apiserver | gawk '{print
$1}')
dcec07c13291
```

Procédez au cryptage de tous les Secrets :

```
root@kubemaster:~/etcd-v3.3.13-linux-amd64# export KUBECONFIG=/etc/kubernetes/admin.conf
root@kubemaster:~/etcd-v3.3.13-linux-amd64# kubectl get secrets --all-namespaces -o json | kubectl replace -f -
secret/default-token-jcvl7 replaced
secret/default-token-z8n7g replaced
secret/default-token-9rdpr replaced
secret/attachdetach-controller-token-7r6jg replaced
secret/bootstrap-signer-token-rknhr replaced
secret/calico-kube-controllers-token-n8crt replaced
secret/calico-node-token-272kx replaced
secret/certificate-controller-token-x26pf replaced
secret/clusterrole-aggregation-controller-token-qxvjj replaced
secret/coredns-token-w74l5 replaced
secret/cronjob-controller-token-8s7fj replaced
secret/daemon-set-controller-token-2mmmg replaced
secret/default-token-8ctj2 replaced
secret/deployment-controller-token-d4fl4 replaced
```

```
secret/disruption-controller-token-wbpcd replaced
secret/endpoint-controller-token-ldcpj replaced
secret/expand-controller-token-wprmr replaced
secret/generic-garbage-collector-token-lssw7 replaced
secret/horizontal-pod-autoscaler-token-nxp94 replaced
secret/job-controller-token-6lnjs replaced
secret/kube-proxy-token-rjg4f replaced
secret/namespace-controller-token-rfd89 replaced
secret/node-controller-token-pb5pq replaced
secret/persistent-volume-binder-token-99sft replaced
secret/pod-garbage-collector-token-cxsjw replaced
secret/pv-protection-controller-token-gkjgf replaced
secret/pvc-protection-controller-token-j6jmq replaced
secret/replicaset-controller-token-xz7mr replaced
secret/replication-controller-token-vhq7f replaced
secret/resourcequota-controller-token-bvgx5 replaced
secret/service-account-controller-token-lkqwb replaced
secret/service-controller-token-bcxft replaced
secret/statefulset-controller-token-9r4j9 replaced
secret/token-cleaner-token-glqqk replaced
secret/ttl-controller-token-bsrvq replaced
```
