

Version - **2024.01**

Dernière mise-à-jour : 2024/12/15 06:55

DOF308 - Introduction à la Sécurisation de K8s

Contenu du Module

- **DOF308 - Introduction à la Sécurisation de K8s**
 - Contenu du Module
 - LAB #1 - Role Based Acces Control et Certificats TLS
 - 1.1 - Présentation
 - 1.2 - Le Fichier /etc/kubernetes/manifests/kube-apiserver.yaml
 - 1.3 - Création d'un serviceAccount
 - 1.4 - Création d'un Utilisateur
 - 1.5 - Certificats TLS
 - LAB #2 - Implémentation de la Sécurité au niveau des Pods
 - 2.1 - Présentation
 - 2.2 - Kubernetes Security Context
 - ReadOnlyRootFilesystem
 - drop
 - 2.3 - Kubernetes Network Policies
 - 2.4 - Kubernetes Resource Allocation Management

Ressources

Lab #1

- <https://www.dropbox.com/scl/fi/ttklc9ejfhpuyq3eh7wbo/flask.yaml?rlkey=gt1fxvfd8a1vxh75e8y8bz6yw&dl=0>
-

- <https://www.dropbox.com/scl/fi/ujzyh5ixqibqtychuyzr/deployment.yaml?rlkey=u4tnbrh2f0b6ewk1mt15y6y63&dl=0>

Lab #2

- <https://www.dropbox.com/scl/fi/sbzoft6ioo6gmo5n56035/readonly.yaml?rlkey=xsqnve5dvkg3l3nbuep06j0tj&dl=0>
- <https://www.dropbox.com/scl/fi/enbctxxwp95s10ssw3l13/drop.yaml?rlkey=pfo8r09cv9zk2xrxohyies9ki&dl=0>
- <https://www.dropbox.com/scl/fi/qptbh81o3gtl8bnii91er/guestbook-all-in-one.yaml?rlkey=5g3cr8a5llggdrme0le254pip&dl=0>
- <https://www.dropbox.com/scl/fi/664obj0d9d0y95kj3czsd/guestbook-network-policy.yaml?rlkey=u3o8yrgpggratq30jgk12rtj90&dl=0>
- <https://www.dropbox.com/scl/fi/f4f3mb8epcy7xr9cgmj1m/flask-resources.yaml?rlkey=l9gptrnet3mh4x5p2v09xvu06&dl=0>

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: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```



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.

Un exemple 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

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

```
root@kubemaster:~# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.56.2:6443
  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
- --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-issuer=https://kubernetes.default.svc.cluster.local
- --service-account-key-file=/etc/kubernetes/pki/sa.pub
- --service-account-signing-key-file=/etc/kubernetes/pki/sa.key
- --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
image: k8s.gcr.io/kube-apiserver:v1.24.2
imagePullPolicy: IfNotPresent
livenessProbe:
  failureThreshold: 8
  httpGet:
    host: 192.168.56.2
```

```
  path: /livez
  port: 6443
  scheme: HTTPS
  initialDelaySeconds: 10
  periodSeconds: 10
  timeoutSeconds: 15
name: kube-apiserver
readinessProbe:
  failureThreshold: 3
  httpGet:
    host: 192.168.56.2
    path: /readyz
    port: 6443
    scheme: HTTPS
  periodSeconds: 1
  timeoutSeconds: 15
resources:
  requests:
    cpu: 250m
startupProbe:
  failureThreshold: 24
  httpGet:
    host: 192.168.56.2
    path: /livez
    port: 6443
    scheme: HTTPS
  initialDelaySeconds: 10
  periodSeconds: 10
  timeoutSeconds: 15
volumeMounts:
- mountPath: /etc/ssl/certs
  name: ca-certs
  readOnly: true
- mountPath: /etc/ca-certificates
```

```
  name: etc-ca-certificates
  readOnly: true
- mountPath: /etc/kubernetes/pki
  name: k8s-certs
  readOnly: true
- mountPath: /usr/local/share/ca-certificates
  name: usr-local-share-ca-certificates
  readOnly: true
- mountPath: /usr/share/ca-certificates
  name: usr-share-ca-certificates
  readOnly: true
hostNetwork: true
priorityClassName: system-node-critical
securityContext:
  seccompProfile:
    type: RuntimeDefault
volumes:
- hostPath:
    path: /etc/ssl/certs
    type: DirectoryOrCreate
  name: ca-certs
- hostPath:
    path: /etc/ca-certificates
    type: DirectoryOrCreate
  name: etc-ca-certificates
- hostPath:
    path: /etc/kubernetes/pki
    type: DirectoryOrCreate
  name: k8s-certs
- hostPath:
    path: /usr/local/share/ca-certificates
    type: DirectoryOrCreate
  name: usr-local-share-ca-certificates
- hostPath:
```

```
path: /usr/share/ca-certificates
type: DirectoryOrCreate
name: usr-share-ca-certificates
status: {}
```

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 dans le Namespace **default** ne peut pas accomplir cette tâche :

```
root@kubemaster:~# 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@kubemaster:~# vi flask.yaml
root@kubemaster:~# 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**.

Appliquez le fichier :

```
root@kubemaster:~# 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.yaml** qui crée des pods qui utiliseront le serviceAccount appelé **flask-backend** :

```
root@kubemaster:~# vi deployment.yaml
root@kubemaster:~# cat deployment.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@kubemaster:~# kubectl create -f deployment.yaml
deployment.apps/myapp-deployment created
```

Vérifiez la présence du deployment :

```
root@kubemaster:~# kubectl get deployment -n flask
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
myapp-deployment    3/3     3             3           32s
```

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

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

1.4 - Création d'un Utilisateur

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

```
root@kubemaster:~# kubectl config get-contexts
CURRENT   NAME                               CLUSTER   AUTHINFO           NAMESPACE
*          kubernetes-admin@kubernetes        kubernetes 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.

En regardant le contexte courant, on voit que l'utilisateur **kubernetes-admin@kubernetes** a deux attributs dénommés :

- client-certificate-data: REDACTED
- client-key-data: REDACTED

```
root@kubemaster:~# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.56.2:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
    name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
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@kubemaster:~# openssl genrsa -out trainee.key 2048  
Generating RSA private key, 2048 bit long modulus  
.....+++  
.....+++  
e is 65537 (0x10001)
```

Créez maintenant un CSR :

```
root@kubemaster:~# 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@kubemaster:~# ls -l /etc/kubernetes/pki/ca.*  
-rw-r--r-- 1 root root 1099 juil. 12 13:23 /etc/kubernetes/pki/ca.crt  
-rw----- 1 root root 1679 juil. 12 13:23 /etc/kubernetes/pki/ca.key
```

Signez donc le CSR :

```
root@kubemaster:~# 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@kubemaster:~# openssl x509 -in trainee.crt -text  
Certificate:  
  Data:  
    Version: 1 (0x0)  
    Serial Number:  
      b6:f7:59:8f:75:19:bc:10  
    Signature Algorithm: sha256WithRSAEncryption  
    Issuer: CN = kubernetes  
    Validity  
      Not Before: Jul 14 07:49:14 2022 GMT  
      Not After : Aug 13 07:49:14 2022 GMT  
    Subject: CN = trainee, O = examplegroup  
    Subject Public Key Info:  
      Public Key Algorithm: rsaEncryption  
      Public-Key: (2048 bit)  
      Modulus:  
        00:9b:2d:e8:7d:ba:e9:9f:b3:da:8f:14:13:21:83:  
        64:c6:6e:7b:2c:ee:4f:e6:71:65:a7:e4:ca:6a:23:  
        ee:cf:e1:43:18:e0:b0:1f:ef:ff:53:21:de:d2:e8:  
        38:d1:39:ab:b0:8d:78:f4:af:7c:80:b0:1a:c3:a2:  
        cb:64:b4:73:e6:a5:30:33:69:f1:6d:9a:5b:66:2e:  
        58:f6:c2:51:7c:42:95:16:ac:60:0e:1d:4d:09:aa:  
        06:29:51:79:f1:45:70:48:b9:1c:e2:05:fc:5c:33:  
        82:d7:82:5f:a2:31:13:b5:23:4c:10:bf:a5:8a:4f:
```

```
37:2a:d6:cc:ac:c7:c0:ad:97:71:95:9e:26:4f:60:
b5:41:8a:7b:c5:79:38:02:28:b0:88:84:23:0b:18:
d2:c2:f9:9f:ff:ec:ec:fb:0a:41:d7:7d:f3:90:2f:
29:08:86:1e:e7:cb:ab:cf:56:5e:a9:ba:06:d8:83:
c2:3c:1d:38:cc:fa:fd:69:17:4e:c3:7e:79:dd:34:
11:9a:ff:5d:32:e4:68:a8:0f:cc:4c:bf:27:bc:2e:
19:b7:9d:ad:68:45:d9:87:06:74:9f:e4:ad:bf:df:
06:c8:28:c7:a4:78:f2:31:b2:6c:c7:9e:90:b8:bf:
48:d4:ae:fd:65:e9:38:fd:8f:30:41:e9:32:f5:de:
69:69
```

Exponent: 65537 (0x10001)

Signature Algorithm: sha256WithRSAEncryption

```
6d:c8:0d:cd:7c:34:5c:08:67:98:b6:ae:80:26:e8:73:f1:14:
3b:02:09:dd:b4:6d:f1:7f:bb:12:8a:16:86:d6:d6:be:ad:92:
99:a8:23:a1:d7:de:d4:e9:03:ec:6f:b9:19:46:2d:d8:f4:30:
71:8c:f0:6e:43:ad:d8:10:46:15:ab:9f:46:c1:56:4c:6c:81:
ab:ba:dd:5b:78:6a:57:82:d3:1a:d7:1a:5f:63:ca:4e:0f:fb:
ce:fe:f1:a5:78:64:a5:03:41:ad:c5:b7:28:45:62:31:ce:02:
09:1b:73:1d:e0:96:a4:1b:c4:09:18:a6:b1:5e:8c:88:03:75:
92:64:47:d3:0c:ce:87:91:9c:25:f7:72:a7:44:9d:36:41:87:
48:61:71:31:9a:24:ae:36:4f:40:c8:f3:08:32:f5:b1:9d:f5:
8a:0a:71:80:e6:70:d9:af:e1:96:55:81:9f:a1:95:39:53:b5:
1b:f3:37:3e:50:d5:a1:6b:d1:4b:d1:c6:75:fb:63:f0:63:06:
ce:99:fb:c3:15:c1:51:3b:ed:d9:c8:68:43:66:3c:ef:92:ba:
ae:a5:0d:02:48:8d:42:1a:70:22:13:75:47:ad:69:d5:48:11:
6b:b1:24:80:7e:d6:0d:f7:92:0c:bb:28:91:6e:d4:4c:a1:14:
c9:2d:47:2c
```

-----BEGIN CERTIFICATE-----

```
MIICujCCAaICCQC291mPdRm8EDANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwpr
dWJlcm5ldGVzMB4XDTEyMDcxNDA3NDkxNFoXDTEyMDgxMzA3NDkxNFowKTEQMA4G
A1UEAwwHdHJhaW5lZTEVMBMGA1UECgwMZXhhbXBsZWdyb3VwMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmY3ofbrpn7PajxQTIYNkxm57L05P5nFlp+TK
aiPuz+FDGOCwH+/UyHe0ug40TmrsI149K98gLAaw6LLZLRz5qUwM2nxbZpbZi5Y
9sJRfEKVFqxdh1NCaoGKVF58UVwSLkc4gX8XD0C14JfojETtSNMEL+lik83KtbM
```

```
rMfArZdxlZ4mT2C1QYp7xXk4AiiwiIQjCxjSwvmf/+zs+wpB133zkC8pCIYe58ur
z1ZeqboG2IPCPB04zPr9aRd0w3553TQRmv9dMuRoqA/MTL8nvC4Zt52taEXZhwZ0
n+Stv98GyCjHpHjyMbJsx56QuL9I1K79Zek4/Y8wQeky9d5paQIDAQABMA0GCSqG
SIb3DQEBCwUAA4IBAQBtyA3NfDRcCGeYtq6AJuhz8RQ7AgndtG3xf7sSihaG1ta+
rZKZqC0h197U6QPsb7kZRI3Y9DBxjPBuQ63YEEYVq59GwVZMbIGrut1beGpXgtMa
1xpfY8p0D/v0/vGleGSlA0GtxbcoRWIxzgIJG3Md4JakG8QJGKaxXoyIA3WSZEfT
DM6HkZwl93KnRJ02QYdIYXExmiSuNk9AyPMIMvWxnfWKCnGA5nDZr+GWVYGfoZU5
U7Ub8zc+UNWha9FL0cZ1+2PwYwb0mfvDFcFR0+3ZyGhDZjzvkrqupQ0CSI1CGnAi
E3VHrWnVSBFrsSSAftYN95IMuyiRbtRMoRTJLUcs
-----END CERTIFICATE-----
```

Créez un deuxième utilisateur dans la même Organisation :

```
root@kubemaster:~# openssl genrsa -out stagiaire.key 2048
Generating RSA private key, 2048 bit long modulus
.....
.....+++
.....+++
e is 65537 (0x10001)

root@kubemaster:~# openssl req -new -key stagiaire.key -out stagiaire.csr -subj "/CN=stagiaire/0=examplegroup"

root@kubemaster:~# 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/0=examplegroup
Getting CA Private Key
```

Créez maintenant le contexte **trainee** :

```
root@kubemaster:~# kubectl config set-credentials trainee --client-certificate=trainee.crt --client-
key=trainee.key
User "trainee" set.
```



```
root@kubemaster:~# kubectl config set-context trainee@kubernetes --cluster=kubernetes --user=trainee
Context "trainee@kubernetes" created.
```

Vérifiez que le contexte soit présent :

```
root@kubemaster:~# kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	kubernetes-admin@kubernetes	kubernetes	kubernetes-admin	
	trainee@kubernetes	kubernetes	trainee	

Utilisez le contexte de trainee :

```
root@kubemaster:~# kubectl config use-context trainee@kubernetes
Switched to context "trainee@kubernetes".
```

```
root@kubemaster:~# kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
	kubernetes-admin@kubernetes	kubernetes	kubernetes-admin	
*	trainee@kubernetes	kubernetes	trainee	

```
root@kubemaster:~# kubectl get pods
Error from server (Forbidden): pods is forbidden: User "trainee" cannot list resource "pods" in API group "" in the namespace "default"
```



Important : Notez que trainee ne peut pas lister les pods parce que les permissions RBAC n'ont pas été définies.

Retournez au contexte de l'administrateur :

```
root@kubemaster:~# kubectl config use-context kubernetes-admin@kubernetes
Switched to context "kubernetes-admin@kubernetes".
```

```
root@kubemaster:~# kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	kubernetes-admin@kubernetes	kubernetes	kubernetes-admin	
	trainee@kubernetes	kubernetes	trainee	

Créez maintenant un **clusterrolebinding** au groupe **examplegroup** :

```
root@kubemaster:~# kubectl create clusterrolebinding examplegroup-admin-binding --clusterrole=cluster-admin --group=examplegroup
clusterrolebinding.rbac.authorization.k8s.io/examplegroup-admin-binding created
```

Utilisez de nouveau le contexte de trainee :

```
root@kubemaster:~# kubectl config use-context trainee@kubernetes
Switched to context "trainee@kubernetes".
```

```
root@kubemaster:~# kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
	kubernetes-admin@kubernetes	kubernetes	kubernetes-admin	
*	trainee@kubernetes	kubernetes	trainee	

```
root@kubemaster:~# kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
calico-kube-controllers-6766647d54-v4hrm	1/1	Running	0	44h
calico-node-5mrjl	1/1	Running	0	41h
calico-node-688lw	1/1	Running	0	44h
calico-node-j25xd	1/1	Running	0	41h
coredns-6d4b75cb6d-dw4ph	1/1	Running	0	44h
coredns-6d4b75cb6d-ms2jm	1/1	Running	0	44h
etcd-kubemaster.ittraining.loc	1/1	Running	1 (44h ago)	44h
kube-apiserver-kubemaster.ittraining.loc	1/1	Running	1 (44h ago)	44h
kube-controller-manager-kubemaster.ittraining.loc	1/1	Running	10 (75m ago)	44h
kube-proxy-bwctz	1/1	Running	0	41h
kube-proxy-j89vg	1/1	Running	0	41h
kube-proxy-jx76x	1/1	Running	0	44h
kube-scheduler-kubemaster.ittraining.loc	1/1	Running	11 (75m ago)	44h

metrics-server-7cb867d5dc-g55k5	1/1	Running	0	28h
---------------------------------	-----	---------	---	-----

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@kubemaster:~# ls -l /var/lib/kubelet/pki/
total 12
-rw----- 1 root root 2851 juil. 12 13:23 kubelet-client-2022-07-12-13-23-12.pem
lrwxrwxrwx 1 root root 59 juil. 12 13:23 kubelet-client-current.pem -> /var/lib/kubelet/pki/kubelet-
client-2022-07-12-13-23-12.pem
-rw-r--r-- 1 root root 2367 juil. 12 13:23 kubelet.crt
-rw----- 1 root root 1675 juil. 12 13:23 kubelet.key
```



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 par 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 privilégié. Les commandes concernées sont **exec** et **attach**.
Un *escalated container* dans un pod privilégié 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 politiques applicables,
- **ValidatingAdmissionWebhooks**,
 - permet d'appeler un service externe qui implémente une politique de sécurité, tel que [Grafefeas](#).

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@kubemaster:~# vi readonly.yaml
root@kubemaster:~# 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@kubemaster:~# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
flask-ro	1/1	Running	0	13m
postgres-deployment-5b8bd66778-j99zz	1/1	Running	7	4d1h
redis-deployment-67d4c466c4-9wzfn	1/1	Running	7	4d1h
result-app-deployment-b8f9dc967-nzbqd	1/1	Running	7	4d1h
result-app-deployment-b8f9dc967-r84k6	1/1	Running	7	3d22h
result-app-deployment-b8f9dc967-zbsk2	1/1	Running	7	3d22h
voting-app-deployment-669dccccfb-jpn6h	1/1	Running	7	4d1h
voting-app-deployment-669dccccfb-ktd7d	1/1	Running	7	3d22h
voting-app-deployment-669dccccfb-x868p	1/1	Running	7	3d22h
worker-app-deployment-559f7749b6-jh86r	1/1	Running	19	4d1h

Connectez-vous au conteneur :

```
root@kubemaster:~# 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/72/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/71/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/70/fs) on / type overlay
```

```
ayfs/snapshots/70/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/69/fs,upperdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/73/fs,workdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/73/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@kubemaster:~# vi drop.yaml
root@kubemaster:~# 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@kubemaster:~# kubectl create -f drop.yaml
pod/flask-cap created
```

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

```
root@kubemaster:~# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
flask-cap	1/1	Running	0	4m4s
flask-ro	1/1	Running	0	13m
postgres-deployment-5b8bd66778-j99zz	1/1	Running	7	4d1h
redis-deployment-67d4c466c4-9wzfn	1/1	Running	7	4d1h
result-app-deployment-b8f9dc967-nzbqd	1/1	Running	7	4d1h
result-app-deployment-b8f9dc967-r84k6	1/1	Running	7	3d22h
result-app-deployment-b8f9dc967-zbsk2	1/1	Running	7	3d22h
voting-app-deployment-669dccccfb-jpn6h	1/1	Running	7	4d1h
voting-app-deployment-669dccccfb-ktd7d	1/1	Running	7	3d22h
voting-app-deployment-669dccccfb-x868p	1/1	Running	7	3d22h
worker-app-deployment-559f7749b6-jh86r	1/1	Running	19	4d1h

Connectez-vous au conteneur :

```
root@kubemaster:~# 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
```

2.3 - Kubernetes Network Policies

Créez le fichier **guestbook-all-in-one.yaml** :

```
root@kubemaster:~# vi guestbook-all-in-one.yaml
root@kubemaster:~# cat guestbook-all-in-one.yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    tier: backend
    role: master
spec:
  ports:
    # the port that this service should serve on
    - port: 6379
      targetPort: 6379
  selector:
    app: redis
    tier: backend
    role: master
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  labels:
```



```
  app: redis
  role: master
  tier: backend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 1
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   app: guestbook
  #   role: master
  #   tier: backend
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
        - name: master
          image: gcr.io/google_containers/redis:e2e # or just image: redis
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          ports:
            - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
```

```
labels:
  app: redis
  tier: backend
  role: slave
spec:
  ports:
    # the port that this service should serve on
    - port: 6379
  selector:
    app: redis
    tier: backend
    role: slave
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-slave
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  labels:
    app: redis
    role: slave
    tier: backend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 2
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   app: guestbook
  #   role: slave
  #   tier: backend
  template:
```

```
  metadata:
    labels:
      app: redis
      role: slave
      tier: backend
  spec:
    containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
          - name: GET_HOSTS_FROM
            value: dns
            # If your cluster config does not include a dns service, then to
            # instead access an environment variable to find the master
            # service's host, comment out the 'value: dns' line above, and
            # uncomment the line below.
            # value: env
        ports:
          - containerPort: 6379
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: frontend
    labels:
      app: guestbook
      tier: frontend
  spec:
    # if your cluster supports it, uncomment the following to automatically create
    # an external load-balanced IP for the frontend service.
```

```
# type: LoadBalancer
ports:
  # the port that this service should serve on
  - port: 80
selector:
  app: guestbook
  tier: frontend
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  # selector can be applied automatically
  # from the labels in the pod template if not set
  # selector:
  #   app: guestbook
  #   tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
```

```

image: corelab/gb-frontend:v5
resources:
  requests:
    cpu: 100m
    memory: 100Mi
env:
- name: GET_HOSTS_FROM
  value: dns
  # If your cluster config does not include a dns service, then to
  # instead access environment variables to find service host
  # info, comment out the 'value: dns' line above, and uncomment the
  # line below.
  # value: env
ports:
- containerPort: 80

```

Installez l'application **Guestbook** :

```
root@kubemaster:~# kubectl create -f guestbook-all-in-one.yaml
```

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
flask-cap	1/1	Running	0	53m	192.168.239.26	kubemaster	<none>
flask-ro	1/1	Running	0	59m	192.168.150.14	kubemaster	<none>
frontend-dhd4w	1/1	Running	0	32m	192.168.150.16	kubemaster	<none>
frontend-dmbbf	1/1	Running	0	32m	192.168.150.17	kubemaster	<none>
frontend-rqr6p	1/1	Running	0	32m	192.168.239.29	kubemaster	<none>

```
<none>
redis-master-zrrr4    1/1    Running    0          32m    192.168.239.27    kubenode1.ittraining.loc    <none>
<none>
redis-slave-jsrt6     1/1    Running    0          32m    192.168.150.15    kubenode2.ittraining.loc    <none>
<none>
redis-slave-rrnx9     1/1    Running    0          32m    192.168.239.28    kubenode1.ittraining.loc    <none>
<none>
...
```

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

```
root@kubemaster:~# kubectl describe pod redis-master-zrrr4 | grep tier
      tier=backend

root@kubemaster:~# kubectl describe pod frontend-dhd4w | 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
```

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

```
root@kubemaster:~# kubectl exec -it redis-master-zrrr4 -- bash
[ root@redis-master-zrrr4:/data ]$
```

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

```
[ root@redis-master-zrrr4:/data ]$ ping -c 4 192.168.150.15
PING 192.168.150.15 (192.168.150.15) 56(84) bytes of data.
64 bytes from 192.168.150.15: icmp_seq=1 ttl=62 time=0.324 ms
64 bytes from 192.168.150.15: icmp_seq=2 ttl=62 time=0.291 ms
64 bytes from 192.168.150.15: icmp_seq=3 ttl=62 time=0.366 ms
64 bytes from 192.168.150.15: icmp_seq=4 ttl=62 time=0.379 ms

--- 192.168.150.15 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3070ms
rtt min/avg/max/mdev = 0.291/0.340/0.379/0.034 ms
```

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

```
[ root@redis-master-zrrr4:/data ]$ ping -c 4 192.168.150.16
PING 192.168.150.16 (192.168.150.16) 56(84) bytes of data.

--- 192.168.150.16 ping statistics ---
```

```
4 packets transmitted, 0 received, 100% packet loss, time 3063ms
```

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

```
[ root@redis-master-zrrr4:/data ]$ exit
exit
command terminated with exit code 1

root@kubemaster:~# kubectl exec -it frontend-dhd4w -- bash
root@frontend-dhd4w:/var/www/html#
```

Installez le paquet **iputils-ping** :

```
root@frontend-dhd4w:/var/www/html# apt update
root@frontend-dhd4w:/var/www/html# apt install iputils-ping -y
```

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

```
root@frontend-dhd4w:/var/www/html# ping -c 4 192.168.150.17
PING 192.168.150.17 (192.168.150.17): 56 data bytes
64 bytes from 192.168.150.17: icmp_seq=0 ttl=63 time=0.185 ms
64 bytes from 192.168.150.17: icmp_seq=1 ttl=63 time=0.112 ms
64 bytes from 192.168.150.17: icmp_seq=2 ttl=63 time=0.093 ms
64 bytes from 192.168.150.17: icmp_seq=3 ttl=63 time=0.121 ms
--- 192.168.150.17 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.093/0.128/0.185/0.035 ms
```

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

```
root@frontend-dhd4w:/var/www/html# ping -c 4 192.168.239.27
PING 192.168.239.27 (192.168.239.27): 56 data bytes
64 bytes from 192.168.239.27: icmp_seq=0 ttl=62 time=0.371 ms
64 bytes from 192.168.239.27: icmp_seq=1 ttl=62 time=0.469 ms
```



```
64 bytes from 192.168.239.27: icmp_seq=2 ttl=62 time=0.349 ms
64 bytes from 192.168.239.27: icmp_seq=3 ttl=62 time=0.358 ms
--- 192.168.239.27 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.349/0.387/0.469/0.048 ms
```

Sortez du pod frontend :

```
root@frontend-dhd4w:/var/www/html# exit
exit
root@kubemaster:~#
```

2.4 - 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   0           67m
flask-resources     1/1     Running   0           53s
flask-ro            1/1     Running   0           74m
...
```

Connectez-vous au pod :

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

Installez le paquet **stress** :

```
root@flask-resources:/# echo "deb http://archive.debian.org/debian/ jessie main contrib non-free" >
/etc/apt/sources.list
root@flask-resources:/# echo "deb http://archive.debian.org/debian-security jessie/updates main contrib non-free"
>> /etc/apt/sources.list
root@flask-resources:/# cat /etc/apt/sources.list
deb http://archive.debian.org/debian/ jessie main contrib non-free
deb http://archive.debian.org/debian-security jessie/updates main contrib non-free
root@flask-resources:/# apt update
root@flask-resources:/# apt install stress -y
```

Testez la limite mise en place :

```
root@flask-resources:/# stress --cpu 1 --io 1 --vm 2 --vm-bytes 800M
stress: info: [41] dispatching hogs: 1 cpu, 1 io, 2 vm, 0 hdd
stress: FAIL: [41] (416) <-- worker 45 got signal 9
stress: WARN: [41] (418) now reaping child worker processes
stress: FAIL: [41] (452) failed run completed in 1s
```

Sortez du pod flask-resources :

```
root@flask-resources:/# exit
exit
root@kubemaster:~#
```