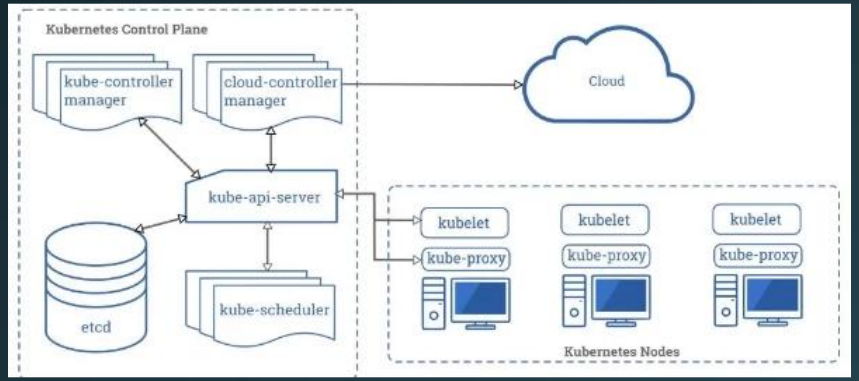


Sécurité Kubernetes

Vue d'ensemble

Protéger vos applications conteneurisées

Le Triangle de la Sécurité K8s

Les trois piliers de la sécurité Kubernetes

La sécurité dans Kubernetes repose sur trois concepts fondamentaux qui forment un triangle de protection complet pour vos applications conteneurisées.

Authentification

Qui êtes-vous ? Identité des utilisateurs et des services.

Mécanismes : Users/ServiceAccounts

Autorisation

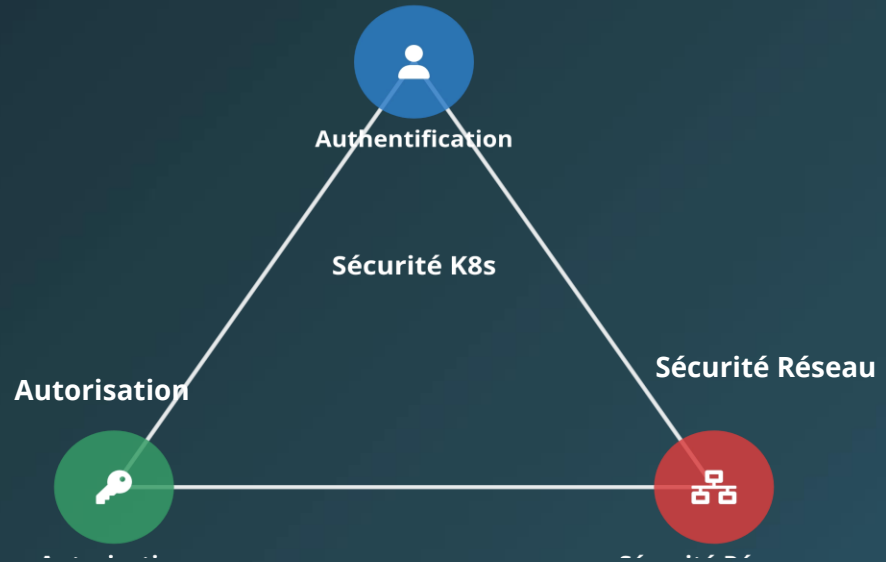
Que pouvez-vous faire ? Permissions accordées.

Mécanismes : RBAC/Rôles

Sécurité Réseau

Où pouvez-vous aller ? Contrôle des communications.

Mécanismes : NetworkPolicies

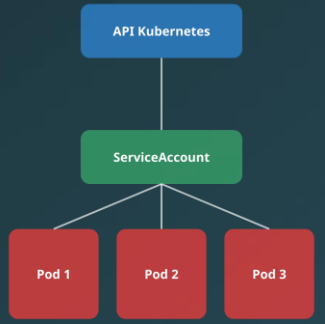


ServiceAccount (SA) - Identité des Pods

Définition

Un **ServiceAccount** est un compte d' **identité** pour les **Pods** (pas pour les humains).

Il permet aux applications s'exécutant dans des conteneurs d'interagir avec l'API Kubernetes de manière sécurisée et contrôlée.



Les Pods utilisent le ServiceAccount comme identité pour interagir avec l'API Kubernetes

Création d'un ServiceAccount

```
apiVersion : v1
Kind : ServiceAccount
Metadata :
  Name : my-app-sa
  Namespace : production
```

Utilisation dans un Pod

```
apiVersion  : v1
kind : Pod
Metadata :
  name :  my-pod
Spec :
  serviceAccountName : my-app-sa      # ⬅️ Identité du Pod
  Containers :
    - Name : app
      Image : nginx
```

RBAC - Contrôle d'Accès Basé sur les Rôles

Qu'est-ce que le RBAC ?

Le **RBAC** (Role-Based Access Control) est un mécanisme d'autorisation qui définit **qui** peut faire **quoi** sur **quelles ressources** dans un cluster Kubernetes.

Il permet de contrôler finement les permissions accordées aux utilisateurs et aux services.

Architecture RBAC

ServiceAccount → **Role** → **Resources**

"Qui" → "Peut quoi" → "Sur quoi"

Composants clés

- **Roles/ClusterRoles** : Définissent les permissions
- **RoleBindings/ClusterRoleBindings:** : Attribuent les rôles aux utilisateurs ou ServiceAccounts
- **Verbs** : Actions autorisées (get, list, create, update, delete, etc.)
- **Resources** : Objets Kubernetes (pods, services, deployments, etc.)

Diagramme RBAC Kubernetes

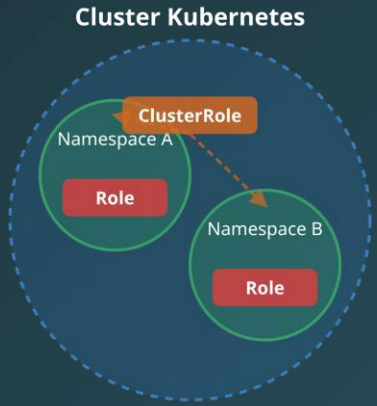
Exemple de flux d'autorisation

Roles & ClusterRoles - Permissions

Role (Namespace-scopé)

Définit des permissions limitées à un **namespace spécifique** .

```yaml
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"]
```



ClusterRole (Cluster-wide)

Définit des permissions s'appliquant à l' **ensemble du cluster** .

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

Comparaison Role vs ClusterRole

Caractéristique	Role	ClusterRole
Portée	Namespace	Cluster entier
Utilisation	Permissions app	Permissions admin
Liaison	RoleBinding	ClusterRoleBinding

RoleBinding & ClusterRoleBinding - Attribution

RoleBinding

Lie un ServiceAccount à un Role dans un namespace spécifique.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: jane # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoleBinding

Lie un ServiceAccount à un ClusterRole à l'échelle du cluster.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "dave" to read secrets in the "development" namespace.
# You need to already have a ClusterRole named "secret-reader".
kind: RoleBinding
metadata:
  name: read-secrets
  #
  # The namespace of the RoleBinding determines where the permissions are granted.
  # This only grants permissions within the "development" namespace.
  namespace: development
subjects:
- kind: User
  name: dave # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

NetworkPolicy - Pare-feu Kubernetes

Concept

Les **NetworkPolicies** contrôlent le **trafic réseau** entre les Pods, agissant comme un pare-feu interne dans votre cluster Kubernetes.

Exemple : "Isoler une base de données"

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: database-isolation
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: database          # ⬅️ CIBLE : Pods database
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: api-server    # ⬅️ AUTORISÉ : Seulement api-server
    ports:
    - protocol: TCP
      port: 5432
```

Points clés des NetworkPolicies

- Contrôlent le trafic **entrant (Ingress) sortant (Egress)**
- Sélectionnent les pods via **podSelector**
- Sont **additives** : plusieurs politiques peuvent s'appliquer
- Nécessitent un **plugin CNI** compatible

Exemple : "Deny all by default"

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector: {}            # ⬅️ Tous les pods
  policyTypes:
  - Ingress
  - Egress
  # Pas de règles = tout est bloqué
```

Bonnes Pratiques de Sécurité

Principe du Moindre Privilège

BON - Permissions minimales

```yaml
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]    # ⬅️ Uniquement le nécessaire
```

MAUVAIS - Permissions trop larges

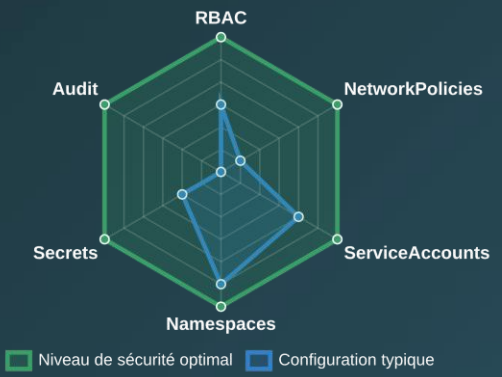
```yaml
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

Segmentation Réseau

Isolation namespace

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-cross-namespace
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Niveau de sécurité



Recommandations clés

- Utilisez des **ServiceAccounts dédiés**
- Implémentez des**NetworkPolicies**
- Appliquez le **principe du moindre privilège**
- Auditez régulièrement les **permissions**

Workflow Sécurisé Complet

Étape 1 : Créer le ServiceAccount

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: secure-app
  namespace: app-ns
```

Étape 2 : Définir les Permissions

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app-ns
  name: app-role
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list"]
```

Étape 3 : Lier les Permissions

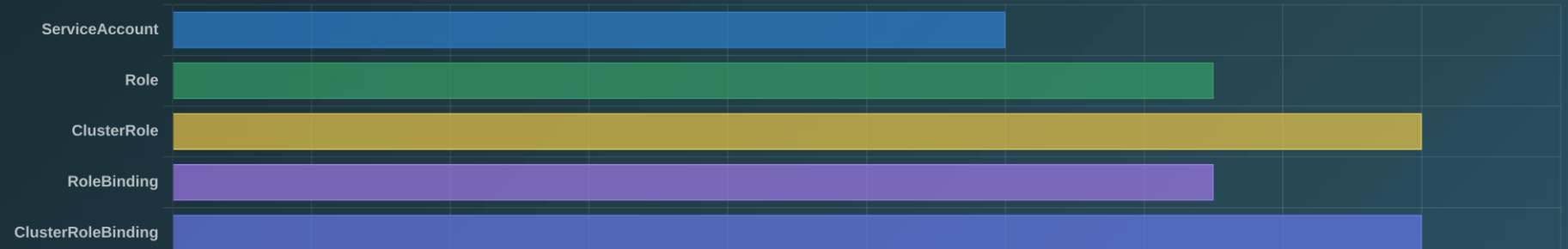
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-binding
  namespace: app-ns
subjects:
- kind: ServiceAccount
  name: secure-app
roleRef:
  kind: Role
  name: app-role
```

Étape 4 : Sécuriser le Réseau

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: app-network-policy
  namespace: app-ns
spec:
  podSelector:
    matchLabels:
      app: secure-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: frontend-ns
```

Résumé des Composants

	Composant	Portée	Objectif
	ServiceAccount	Namespace	Fournir une identité aux Pods pour interagir avec l'API Kubernetes
	Role	Namespace	Définir des permissions limitées à un namespace spécifique
	ClusterRole	Cluster	Définir des permissions s'appliquant à l'ensemble du cluster
	RoleBinding	Namespace	Lier un ServiceAccount à un Role dans un namespace
	ClusterRoleBinding	Cluster	Lier un ServiceAccount à un ClusterRole à l'échelle du cluster
	NetworkPolicy	Namespace	Contrôler le trafic réseau entre les Pods (pare-feu interne)



Commandes de Vérification

Vérifier les permissions d'un ServiceAccount

```
$ kubectl auth can-i get pods --as=system:serviceaccount:production:my-app-sa
yes
```

Lister les NetworkPolicies

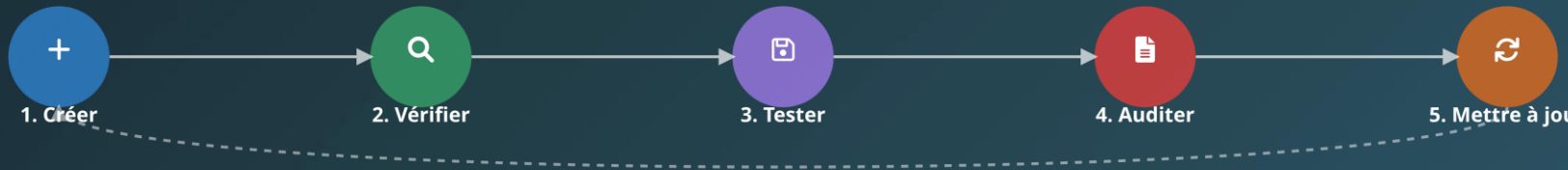
```
$ kubectl get networkpolicies --all-namespaces
NAMESPACE NAME POD-SELECTOR AGE
production database-isolation app=database 24h
```

Vérifier les RoleBindings

```
$ kubectl get rolebindings,clusterrolebindings -n production
NAME ROLE AGE
read-pods Role/pod-reader 24h
```

Audit des permissions

```
$ kubectl describe role pod-reader -n production
Name: pod-reader
PolicyRule:
Resources Non-Resource URLs Resource Names Verbs
--------- ----------------- ------------ -----
pods [] [] [get list watch]
```



La sécurité dans Kubernetes repose sur le principe du "least privilege" : donner uniquement les permissions nécessaires !

Conclusion

Sécurité multicouche dans Kubernetes

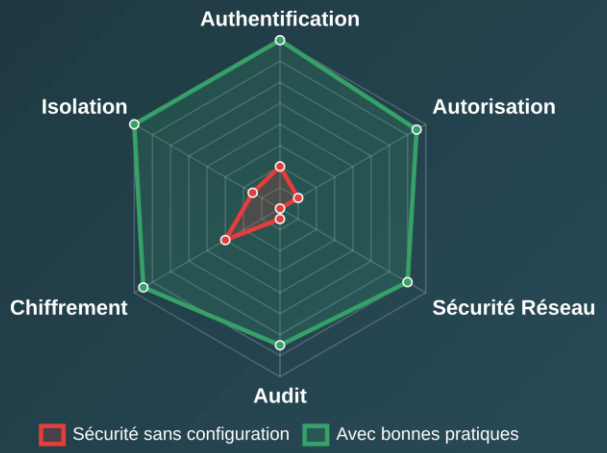
Authentification
Identité des utilisateurs et des services via ServiceAccounts

Autorisation
Contrôle d'accès précis via RBAC, Roles et RoleBindings

Sécurité Réseau
Isolation et contrôle du trafic via NetworkPolicies

Principe du moindre privilège
Accorder uniquement les permissions nécessaires

Défense en profondeur



Questions ?

Merci pour votre attention

kubernetes.io/docs/concepts/security

Kubernetes Security