NETWORKPOLICY (FIREWALL INTERNE DES PODS KUBERNETES)

Introduction

Un NetworkPolicy est une spécification de **la façon dont les pods sont autorisés à communiquer entre eux**, c'est en quelque sorte un Firewall interne au pod qui **autorise ou interdit les connexions entrantes ou/et sortantes**. Le NetworkPolicy utilise des labels pour sélectionner les pods et définir des règles qui spécifient le trafic autorisé vers les pods sélectionnés.

Les stratégies réseau sont implémentées par le plugin réseau (CNI, rappel ici), vous devez donc utiliser une solution réseau qui prend en charge les NetworkPolicies. Par exemple, au jour ou j'écris cet article, le CNI flannel ne supporte pas les NetworkPolicies.

> **Information**
>
> Par défaut, les pods ne sont pas isolés, c'est-à-dire qu'ils acceptent le trafic de toute source. Les pods deviennent **isolés** en ayant un NetworkPolicy.

NetworkPolicy

Ingress et Egress

Avant de commencer par la création d'un NetworkPolicy, il est important au préalable de définir le mot Ingress et Egress :

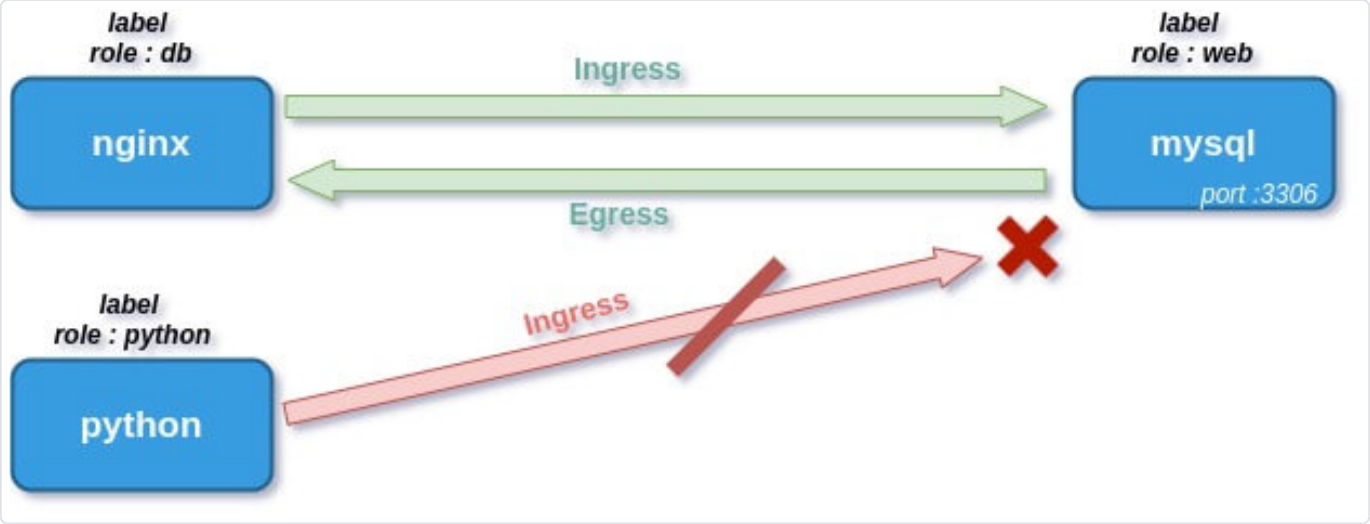
- **Ingress**: traffic entrant sur le pod
- **Egress**: traffic sortant sur le pod

Si vous souhaitez limiter les communications entrantes de votre pod, alors vous utiliserez alors une règle Ingress, dans le cas ou vous souhaitez contrôler les communications sortantes de notre pod on utilisera des règles Egress.

Exemple

Dans cet exemple, on souhaite **protéger les communications des pods de notre cluster** . Dans ce cluster, nous disposons d'un pod de base de données mysql avec comme label `role: db` écoutant sur le port 3306, et un second pod web nginx écoutant sur le port 80 avec comme label `role: web`.

Dans un plan de sécurité continue, nous ambitionnons de **sécuriser la communication** entre nos deux pods, en n'autorisant que le traffic provenant de notre pod web nginx avec le role `role: web` vers notre pod mysql. Nous allons pour notre exemple, créer un NetworkPolicy avec une règle Ingress sur notre pod mysql. Ce qui nous donne le schéma suivant :



Afin de réaliser notre exemple, nous commencerons par créer nos deux pods. Débutons avec le pod de base de données mysql ayant le label `role: db` :

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: db
  name: mysql
spec:
  containers:
  - image: mysql
    name: mysql
```

Lançons la commande de création :

```
kubectl create -f mysql.yaml
```

Ensuite, nous créons un pod de notre serveur web avec le label `role: web` :

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    role: web
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Exécutons la commande de création :

```
kubectl create -f web.yaml
```

Maintenant, nous passerons à la partie la plus intéressante, à savoir la **création de notre NetworkPolicy**. Comme dit précédemment, nous créerons une règle Ingress sur notre pod mysql n'autorisant que le traffic provenant de notre pod nginx. Pour ce faire, nous nous baserons sur les labels de chaque pod :

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: web
    ports:
    - protocol: TCP
      port: 3306
```

- **`podSelector`** : chaque NetworkPolicy inclut un podSelector qui sélectionne le groupe de pods auquel la politique s'applique. L'exemple ci-dessus sélectionne des modules avec la label **`role = db`**. Si le champ est vide alors le podSelector sélectionne tous les pods de le namespace.
- **`policyTypes`** : chaque NetworkPolicy comprend une policyTypes qui peut inclure soit la valeur **`Ingress`**, **`Egress`** ou les deux. Ce champ indique si la stratégie donnée s'applique ou non au trafic entrant vers le pod sélectionné, au trafic sortant des pods sélectionnés, ou les deux. Si aucun policyTypes n'est spécifié sur un NetworkPolicy, par défaut, l'Ingress sera toujours défini.
- **`Ingress`** : chaque NetworkPolicy peut inclure une liste de règles ingress. Chaque règle autorise un trafic qui correspond à la fois aux sections **`from`** et **`ports`**. L'exemple de stratégie contient une règle unique sur le port 3306, autorisant tous les pods avec le label **`role: db`** sur le namespace **`default`**. (Vous pouvez aussi autoriser des plages d'ip avec le champ **`ipBlock`**, ou par namespace **`namespaceSelector`**)

Il suffit maintenant de créer votre NetworkPolicy avec la commande `kubectl apply ou create` :

```
kubectl create -f mysql.yaml
```

Conclusion

Dans ce chapitre, nous avons pu **comprendre le fonctionnement des NetworkPolicies dans kubernetes**. Vous l'aurez compris, les NetworkPolicies sont un bon moyen pour protéger vos pods contre les communications inutiles voire dangereuses. Par la suite, nous nous sommes consacrés à un exemple pratique, mais si jamais vous souhaitez vous exercer davantage, essayez alors de créer une règle Egress en plus de la règle Ingress, n'autorisant que du traffic sortant vers vos pods nginx depuis votre pod mysql.